

Discovering Temporal Structure: An Overview of Hierarchical Reinforcement Learning

Martin Klissarov*

Mila, McGill University

MARTIN.KLISSAROV@MAIL.MCGILL.CA

Akhil Bagaria*†

Amazon

AKHILBG@AMAZON.COM

Ziyan “Ray” Luo*

Mila, McGill University

ZIYAN.LUO@MAIL.MCGILL.CA

George Konidaris†

Brown University

GDK@CS.BROWN.EDU

Doina Precup†

*Mila, McGill University
Canada CIFAR AI Chair*

DPRECUP@CS.MCGILL.CA

Marlos C. Machado†

*Amii, University of Alberta
Canada CIFAR AI Chair*

MACHADO@UALBERTA.CA

Abstract

Developing agents capable of exploring, planning, and learning in complex environments is a grand challenge in artificial intelligence (AI). Hierarchical reinforcement learning (HRL) offers a promising solution to this challenge by discovering and exploiting temporal structure within an environment, via constructs such as options, skills, or goal-conditioned policies. The strong appeal of the HRL framework has led to a rich and diverse body of literature, but it is still not clear what constitutes good structure, or the kind of problems in which identifying it may be helpful. We identify the broad classes of benefits—and resulting tradeoffs—that HRL may bring to an agent, and use them to introduce and organize methods for discovering temporal structure, in three settings: learning directly from online experience, learning using offline datasets, and leveraging foundation models. Finally, we highlight the remaining challenges of temporal structure discovery and discuss domains in which HRL is particularly well-suited.

*. Equal contribution.

†. Work done while at Brown University.

‡. Equal supervision.

Table of Contents

1	Introduction	4
2	What is Hierarchical Reinforcement Learning For?	5
2.1	The Benefits of Hierarchical Reinforcement Learning	6
2.2	Trade-offs	8
3	Formalizing Hierarchical Reinforcement Learning	10
3.1	Reinforcement Learning	10
3.2	Hierarchical Reinforcement Learning	12
3.2.1	Options: A Mathematical Formalism	12
3.2.2	Option Discovery	15
3.2.3	Related Terminologies and Formalisms	19
3.2.4	Beyond Architectural Choices	21
4	Discovery from Online Experience	21
4.1	Bottleneck Discovery	22
4.2	Spectral Methods	27
4.3	Sequentially Composable Options	32
4.4	Empowerment Maximization	37
4.5	Via Environment Rewards	43
4.6	Curriculum and Hindsight Learning	47
4.7	Meta Learning	52
4.8	Intrinsic Motivation	56
4.9	Directly Optimizing for the Benefits of Hierarchical Reinforcement Learning	61
5	Discovery through Offline Datasets	65
5.1	Variational Inference of Skill Latents	66
5.2	Hindsight Subgoal Relabeling	71
6	Discovery with Foundation Models	75
6.1	Embedding Similarity	76
6.2	Providing Feedback	80
6.3	Reward as Code	83
6.4	Directly Modeling the Policy	85
7	Using Discovered Options	88
7.1	When to Deliberate	88
7.1.1	Call-and-return	89
7.1.2	Interruptions	89
7.2	How to Select Actions	90
7.2.1	Single Option	90
7.2.2	Option Composition	90
7.3	Learning High-level Policies	91
7.3.1	Model-free Approaches	91

7.3.2	Model-based Approaches	92
7.3.3	In-context Learning	94
8	Challenges of Discovery	94
8.1	Non-Stationarity	95
8.2	Simultaneously Learning About Multiple Behaviours	96
8.3	Combining Rewards	97
9	Related Fields	97
9.1	State and Action Abstractions in Reinforcement Learning	97
9.2	Continual Reinforcement Learning	99
9.3	Programmatic Reinforcement Learning	100
9.4	Cooperative Multi-Agent Reinforcement Learning	102
10	Promising Domains for Hierarchical Reinforcement Learning	102
11	Conclusion	104

1. Introduction

Reinforcement learning (RL) is a general computational framework for designing agents that learn to maximize a scalar reward from experience. RL agents sense their environment and produce actions at every single timestep, but effective reward maximization in complex environments requires reasoning and learning over many timescales. Consider how we typically go about our day: we actuate muscles every few milliseconds, while simultaneously making high-level decisions such as choosing presents for a loved one, deciding what to eat for lunch, and planning the next step in our career. Such abstract decision-making allows us to decide how to act in a complex world without being overwhelmed by unnecessary detail.

Hierarchical reinforcement learning (HRL) formalizes the idea of flexibly reasoning over different timescales by developing agents that learn, predict, and act in the world at multiple levels of abstraction. At its core, HRL builds on the temporal structure discovered through interaction with an environment. When that temporal structure is defined by a human expert, HRL can dramatically ease the decision-making burden of the agent by improving exploration (Bellemare et al., 2020), learning (Vinyals et al., 2019), and generalization (Ahn et al., 2022). On the other hand, when it is misaligned with the environment, it can hamper learning—for example, resulting in pathologically bad exploration (Jong et al., 2008). These appeals and drawbacks naturally lead to the question: how can agents autonomously discover useful temporal structure in HRL?

Before designing algorithms that successfully address the discovery problem, we are faced with the question of what constitutes a “good” temporal structure in the first place. Is there one type of “good” structure that yields higher rewards in all possible environments? Are there specific types of problem settings, like multi-task learning (Plappert et al., 2018) and continual learning (Khetarpal et al., 2022), where we expect HRL to outperform non-hierarchical RL, and others where we do not? How can prior knowledge, obtained for instance via the integration of large language models (LLMs), alleviate the difficulty of discovery? This work presents various perspectives on what constitutes “good” temporal structures through the lens of the fundamental problems of RL—specifically, how HRL can aid **exploration**, **credit assignment**, **transfer**, and **interpretability**.

Key Contributions. The discovery of useful temporal structures has been a prolific, albeit challenging, topic of research. Before we present the various algorithms that tackle this problem, we take a step back and discuss the potential **benefits** of HRL methods as well as their **trade-offs** in the context of sequential decision-making. It is through the lens of these benefits and trade-offs that we introduce the diverse approaches that have been developed to tackle the fundamental question of discovery. While recent surveys in HRL (Pateria et al., 2021; Hutsebaut-Buyse et al., 2022) present papers based on their technical differences and domains of application, we present the literature based on how each method contributes to these core benefits. We then discuss the **challenges** associated with discovering structure in HRL and the **domains** that are particularly well-suited for such methods.

Scope. We study temporal structure discovery as a problem faced by the agent. As shown in Figure 1, we categorize approaches in terms of the source of data and prior knowledge, presenting works that (1) discover temporal structure from the agent’s **online experience** (Section 4), (2) leverage **offline datasets** (Section 5), and (3) build on **foundation models**

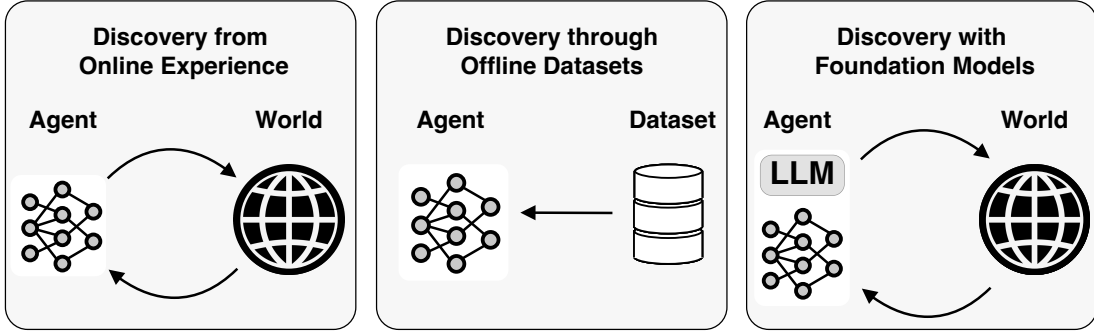


Figure 1: **Types of methods for temporal structure discovery.** We focus on discovering temporal structure autonomously from data under different assumptions.

such as LLMs (Section 6). Most of these algorithms can be instantiated with deep neural networks. For each method within these classes, we discuss the major benefits and limitations that motivate future research. We also study how to leverage temporal structure in decision-making (Section 7), whether discovered or pre-existing.

2. What is Hierarchical Reinforcement Learning For?

HRL aims to exploit the temporal structure of sequential decision-making problems. Solutions to complex tasks can often be deconstructed into simpler subtasks that are modular and composable. **Modularity** refers to the property that a solution to a subtask can be reused without concern for exactly how it was solved. **Compositionality** means that solutions to subtasks can subsequently be recombined to solve a wide range of more complex problems.

To better understand what such a structure might represent in practice, consider a programmer with an abundance of time who cares about solving only a single task. In such a scenario, assembly language might be the optimal choice because its precise control over hardware resources maximizes memory efficiency and minimizes execution time. However, in practice, programmers often opt for higher-level programming languages and software libraries because they offer modules that can be composed to solve common programming subtasks, making writing new programs efficient at the cost of increased execution time. In fact, most large software projects would simply be infeasible without such programming languages. This idea is visually represented in Figure 2, where abstract interfaces allow us to manipulate machine language efficiently. Modularity and compositionality are also particularly appealing properties for software expected to undergo changes throughout its life cycle, because modules can be updated and debugged locally, without considering the global complexity of the task they are used to solve. Developing such a library requires careful consideration of the right code organization to adopt and which guiding principles to follow, balancing execution time, readability, and performance. However, once a library is written, the user can focus on the overall program’s behaviour without needing to understand the implementation details of each function (Wilkes et al., 1951). This is also represented in Figure 2, where the modular and compositional nature of PyTorch allows researchers to efficiently explore research ideas.

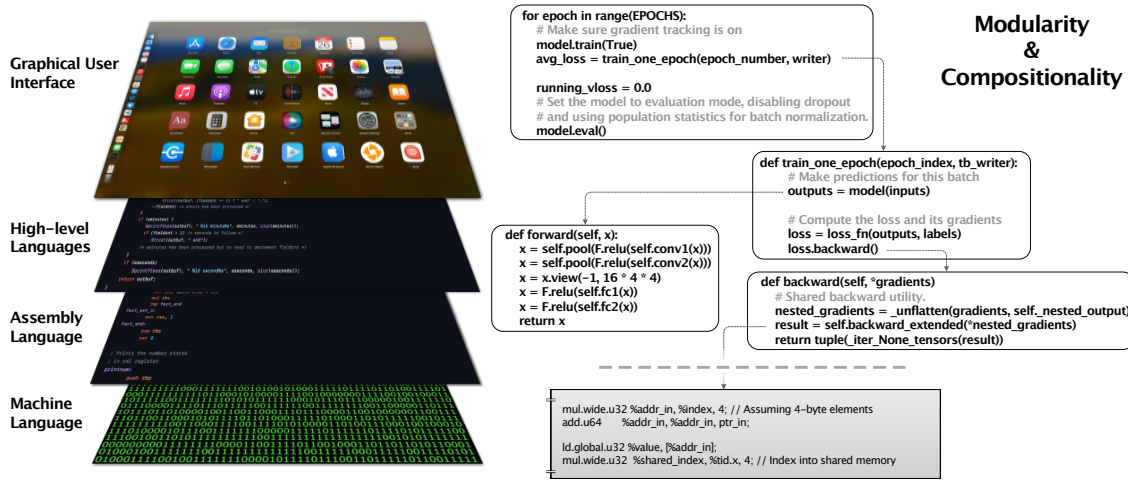


Figure 2: **(Left)** The hierarchy of software and languages which make computers especially useful, allowing humans to directly interact with a user interface or a high-level language to achieve a diversity of goals. **(Right)** a code snippet, revealing the usefulness of modular and compositional software structures that researchers use every day. The PyTorch language, which abstracts over the PTX language used within CUDA kernels, allows exploring research ideas efficiently.

The temporal structure at the core of HRL is analogous to functions and subroutines in programming languages.¹ Just as a human programmer must write a complex program by breaking it into subtasks, so must RL agents autonomously identify hierarchical structure in a stream of data. Modularity and compositionality properties are therefore good indicators as to the kind of problems in which we might find HRL particularly useful (see Section 10). We now discuss how, by discovering and leveraging such temporal structure, HRL methods can help address fundamental challenges in decision-making.

2.1 The Benefits of Hierarchical Reinforcement Learning

Just as modular and compositional codebases can facilitate effective software development, HRL can leverage an environment’s structure to improve decision-making. This is particularly powerful when an agent is faced with complex, long-horizon tasks that can be broken into manageable subtasks. To understand this more precisely, we describe the benefits of HRL through the lens of three fundamental challenges agents face when learning from interaction: how to select the right data to collect (*exploration*), how to efficiently learn from this data (*credit assignment*), and how to transfer knowledge to new situations (*transferability*). Additionally, as agents become increasingly more capable, a new challenge emerges: understanding their decision-making processes (*interpretability*). When covering the different families of HRL methods (Sections 4, 5, and 6), we will explicitly consider how these benefits are instantiated in practice.

1. We provide more details about this analogy in Section 9.3.

Exploration. Broadly speaking, RL agents must solve two problems: (a) how to use existing data to learn useful behaviours, and (b) what data to collect in the first place. The latter, known as the *exploration* problem, is both unique and central to RL; the agent must learn how to collect data that improves its understanding of the world even if doing so does not immediately maximize reward in the short term (Amin et al., 2021). By exploiting the temporal structure of an environment, an agent can improve exploration in at least three ways. First, it can *target states that facilitate progress*, for example by defining subgoals that are more immediately attainable than the overall task and can be organized into a progression resembling a curriculum. Second, it can *explore over extended time scales* by committing to temporally coherent behaviours, rather than selecting individual actions at each step. Finally, agents can *explore at a higher level of abstraction* than individual actions, enabling them to search the solution space more efficiently. Consider a researcher tackling an important scientific question. By learning a high-level programming language, such as PyTorch, and writing modular code, the researcher can iterate faster to investigate many high-level ideas. When iterating over ideas, the researcher may seek to achieve some important milestones, such as constructing a minimal prototype, that can reveal new perspectives and provide insights into possible future courses of action.

Credit Assignment. To improve its decisions over time, an agent must identify the key moments in a sequence of decisions that best explain the observed result. RL algorithms typically leverage multi-step error propagation (Sutton, 1988) to learn about temporally distant, or *delayed*, outcomes. An agent leveraging the environment’s temporal structure could more efficiently identify the origins of an outcome by propagating errors at the abstraction level defined by this structure. Consider our previous example of a researcher performing a scientific experiment. Completing such an experiment consists of a sequence of high-level decisions, such as the choices of data preprocessing or evaluation metrics. Each high-level decision is instantiated through a series of keystrokes that make up the final working code. By reflecting on the validity of the sequence of high-level decisions, rather than of each individual keystroke, the researcher can better identify which ones were critical for the observed outcome. By breaking down a task into such segments, it is also easier to identify if a particular segment is completed, narrowing down the search for lower-level mistakes, such as an errant keystroke that introduced a bug.

Transfer. HRL offers a particularly promising way of exploiting structure shared across different situations: temporally extended skills acquired in one context can be reused and adapted in others (Thrun and Schwartz, 1994). Consider our previous example of an AI researcher conducting experiments and writing a paper for a particular conference. The reusable code and writing skills developed while working on one paper could substantially reduce the effort required to produce a follow-up article, improving the agent’s ability to conduct high-quality research. Such transfer can extend beyond behaviours to include representations, models, and other forms of knowledge. More broadly, transferring skills between distinct tasks is one instance of the broader goal of *generalization*: the ability to apply learned knowledge in situations beyond those encountered during learning. For example, with suitable representations, options can remain useful under similar but previously unseen conditions (Ravindran and Barto, 2003; Konidaris and Barto, 2007). When an agent must continually adapt to a non-stationary environment, explicit task boundaries may be absent, yet temporal abstractions can still support generalization across changing contexts (see Section 9.2).

By addressing these three fundamental challenges, **HRL can achieve faster learning**. Beyond them, HRL offers two further benefits of a different character: *planning*, which is mutually reinforcing with the three rather than independent of them, and *interpretability*, which concerns whether humans can understand what or how the agent has learned.

Planning. The three benefits above concern how an agent learns from experience. But experience is costly—real interaction takes time and can carry risk—and the environment may change over time. Both problems can be mitigated by substituting computation for interaction. This is *planning*: producing or improving decisions by computing with a model of the environment (Sutton and Barto, 2018, Chapter 8); for example, by learning from simulated experience or by dynamic programming. Temporal abstraction makes this substitution effective over long horizons: models of temporally extended behaviours cover the same horizon in fewer applications, thereby reducing the accumulation of model errors. Planning is not a fourth independent benefit but is synergetic with the other three: exploration, credit assignment, and transfer allow an agent to construct plannable abstract models, and planning with those models improves each of them in turn—focusing exploration on the most useful parts of the state space, propagating credit in temporally-extended jumps, and reusing options and their models across similar tasks.

Interpretability. While not all HRL algorithms produce interpretable behaviour, those that do can help human observers better understand an agent’s decision-making process. As agents become increasingly powerful they will be deployed in real-world situations where the consequences of their actions carry considerable stakes, and interpreting their decisions will be a key aspect of ensuring their alignment (Amodei et al., 2016). HRL could provide an interpretable interface for AI alignment via a more abstract decision formulation than the one defined in the environment. For instance, in a robot navigation task, low-level actions such as the forces applied at the joints are particularly hard to interpret compared to a sequence of semantically meaningful skills such as “reach the stairs” and “descend to the first floor”. Humans may follow the agent’s reasoning at that level of abstraction and provide corrective feedback.

2.2 Trade-offs

HRL builds on the inductive bias that tasks can be naturally decomposed into simpler subtasks, making it especially effective when such a hierarchical structure is apparent. This assumption does not imply that a suitable decomposition is known in advance: the structure may be predefined using domain knowledge, or it may need to be discovered from experience. When a suitable hierarchy is provided, HRL can exploit it to improve learning or planning efficiency (Fruit and Lazaric, 2017; Wen et al., 2020). However, consistent with the intuition behind the No Free Lunch Theorem (Wolpert and Macready, 1997), the benefits of this inductive bias depend on its alignment with the problem. A poorly matched decomposition can slow learning or lead to suboptimal policies (Jong et al., 2008). Taking the analogy of programming, adding abstractions in a codebase can simultaneously help in seeing a broader picture, but also obscure the important details (Victor, 2011). When a suitable hierarchy is not provided, the temporal structure itself must be *discovered* by the HRL agent. This discovery incurs a cost in computation and potentially additional environment interactions. The central question is therefore whether, and over what horizon, the benefits of exploiting

the learned hierarchy amortize the cost of discovering it. HRL must therefore be assessed in terms of both the suitability of its hierarchy and the amortization of discovery costs, with trade-offs among *performance*, *sample efficiency*, and *computational efficiency*, as illustrated in Figure 3.

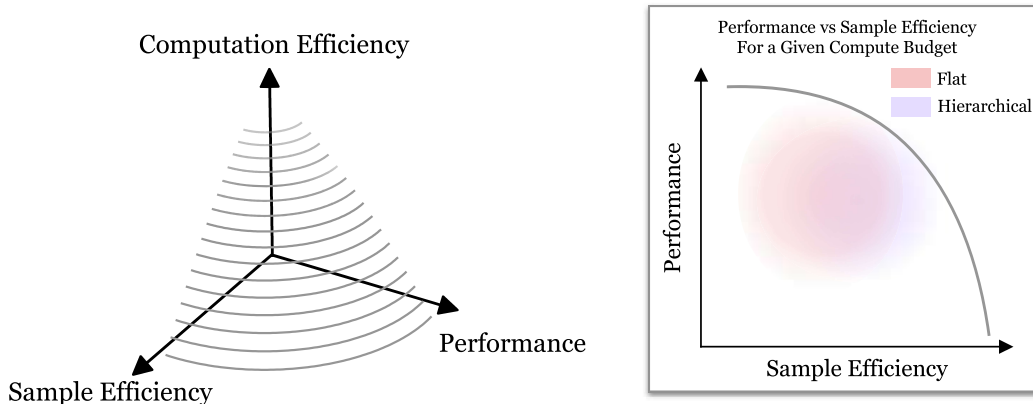


Figure 3: **(Left)** Agents trade-off three different objectives: performance (in terms of reward), sample efficiency (the amount of data required), and computational efficiency (amount of computation required). The Pareto frontier between performance and sample efficiency shifts with different compute budgets. For instance, with unlimited compute (a low computational efficiency), high performance can be achieved at moderate sample efficiency. **(Right)** A qualitative illustration of a “flat” agent and a hierarchical agent under the Pareto frontier of the performance vs. sample efficiency trade-off, given a fixed compute budget. A “flat” agent, as opposed to a hierarchical one, does not use temporally-extended actions. HRL agents typically trade some amount of performance for improved sample efficiency.

We first consider the trade-off between performance and sample efficiency. This trade-off can be appreciated from a theoretical point of view: the optimal policy can always be represented using primitive actions alone (Bertsekas, 1995), and restricting the agent to a predefined set of options may exclude optimal behavior (Jong et al., 2008). However, learning the optimal policy for large and realistic environments is often intractable. Rather than pursuing perfect solutions, an agent could embrace efficient learning algorithms that find reasonable but often suboptimal policies. This is one of the main appeals of HRL: by re-composing existing solutions (i.e., behaviours achieving particular subtasks), an agent may be able to quickly find approximate solutions for a variety of tasks. For example, a pre-trained skill that opens doors allows an agent to bypass learning the complex motor controls for that specific action. However, this very abstraction can be limiting; if a door is stuck and requires an unusual push-and-jiggle motion, the rigid pre-defined skill might fail, preventing the agent from solving an edge case that a more flexible, low-level policy could have.

Another important challenge faced by agents interacting with complex and realistic environments is computational efficiency: the total compute needed to reach a target performance within a given sample budget. The compute includes learning, planning, action selection, and, when needed, discovering temporal abstractions. These computational demands interact with both performance and sample efficiency: an agent may extract more

value from limited experience by spending additional computation on learning reusable behaviours or planning with a learned model, whereas reducing computation may compromise performance or increase the need for further interactions. Temporal abstraction can improve this balance by shortening the effective planning horizon. For example, a robot equipped with an LLM may plan over semantically meaningful skills rather than directly search over sequences of continuous motor commands. When these skills match the task, the agent can exploit their internal behaviours without repeatedly reasoning through low-level details. Planning with temporally extended actions can consequently require far fewer iterations to reach a near-optimal solution (Mann et al., 2015), with efficiency gains depending on the structure exploited by the hierarchy (Wen et al., 2020). These savings, however, must be weighed against the costs of constructing and using the hierarchy. Selecting a skill may require deliberation across multiple levels, with costs depending on neural network size, the number and depth of Monte Carlo Tree Search rollouts (Coulom, 2006), or the length of an LLM’s reasoning trace. Executing a skill over multiple timesteps amortizes this cost, but may compromise performance when changing conditions require a different behaviour; more frequent deliberation can improve responsiveness at a higher computational cost. Discovering and refining the hierarchy likewise requires computation and may require additional samples, although existing experience can also be reused. Thus, HRL’s three-way trade-off concerns how much computation and experience to invest in constructing and using a hierarchy to attain the desired performance, and whether its subsequent benefits amortize that investment.

On the Importance of Prior Knowledge Reuse. A common pitfall in HRL applications is that: *discovering* a problem’s hierarchical structure (see Section 3.2.2) can require more samples and computation than solving the problem directly using a “flat” agent, highlighting the importance of carefully choosing when to apply HRL (see Section 10). This cost can be amortized in different ways. For example, it may be offset if the agent is expected to complete *many different tasks within its lifetime*, allowing the learned subtasks to be potentially reused. Alternatively, reusing existing knowledge—such as **offline datasets** (Section 5) and **foundation models** (Section 6)—can help mitigate this cost. As we will see, HRL offers a natural and particularly promising way to incorporate such prior knowledge.

3. Formalizing Hierarchical Reinforcement Learning

We use the notation introduced by Sutton and Barto (2018): capital letters refer to random variables, whereas lowercase letters refer to their instantiation. Table 2 summarizes the notation used in this section.

3.1 Reinforcement Learning

We consider an agent interacting with an environment where the agent is in state $S_t \in \mathcal{S}$ at timestep t , selects an action $A_t \in \mathcal{A}$, and in response the environment emits a scalar reward $R_{t+1} \in \mathbb{R}$ and transitions to a new state, $S_{t+1} \in \mathcal{S}$. This transition happens according to a transition probability distribution,

$$p(s'|s, a) = p(S_{t+1} = s' | S_t = s, A_t = a). \quad (1)$$

The agent’s goal is to find a policy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$, where $\Delta(\mathcal{A})$ is the distribution over $\mathcal{A}$, that maximizes the expected discounted sum of rewards (return):

$$G_t = \sum_{i=t}^{\infty} \gamma^{i-t} R_{i+1}, \quad (2)$$

where $\gamma \in [0, 1)$ is the discount factor. This 5-tuple, $\langle \mathcal{S}, \mathcal{A}, R, p, \gamma \rangle$, defines a Markov Decision Process (MDP) (Puterman, 1994), the most commonly accepted formalism in RL.

When following a particular policy π , the value of each state can be represented by the state value function,

$$v_{\pi}(s) = \mathbb{E}_{\pi} [G_t | S_t = s]. \quad (3)$$

Similarly, we may consider the value of being in state s and taking action a , following policy π afterward, represented by the action-value function, or q -function,

$$q_{\pi}(s, a) = \mathbb{E}_{\pi} [G_t | S_t = s, A_t = a]. \quad (4)$$

This function can be written recursively,

$$\begin{aligned} q_{\pi}(s, a) &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s, A_t = a, \pi] \\ &= r(s, a) + \gamma \sum_{s'} p(s' | s, a) v_{\pi}(s') \\ &= r(s, a) + \gamma \sum_{s'} p(s' | s, a) \sum_{a'} \pi(a' | s') q_{\pi}(s', a'). \end{aligned} \quad (5)$$

A similar derivation is possible for the value function, and this set of equations is referred to as the Bellman equations for evaluation (Bellman, 1957).

The goal of an RL agent is to maximize the rewards it gets from interacting with the environment. In an MDP, there exists at least one optimal policy, defined as,

$$\pi^* = \arg \max_{\pi} q_{\pi}(s, a). \quad (6)$$

In most practical settings, where the state or action spaces are large or continuous, the transition dynamics are unknown, or only sampled experience is available, this quantity is impractical to compute exactly, necessitating function approximation methods. Such approximations stem from two families of algorithms for learning reward-maximizing policies. The first family of methods, called value-based methods, greedily maximizes an estimated action-value function. Q-Learning (Watkins and Dayan, 1992) is likely the most used algorithm to estimate the optimal policy, π^* , whose update rule takes the following form,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right]. \quad (7)$$

This update has been the basis of the Deep Q-Networks (DQN) algorithm (Mnih et al., 2015).

The second family of methods directly maximizes the quantity of interest, that is, the discounted sum of returns. The policy gradient theorem (Sutton et al., 1999a) provides the

gradient of the expected discounted return from an initial state distribution, $d(s_0)$, with respect to a stochastic policy, $\pi_\zeta(\cdot|s)$, parameterized by ζ ,

$$\frac{\partial J(\zeta)}{\partial \zeta} = \sum_s d_\pi^\gamma(s) \sum_a \frac{\partial \pi_\zeta(a|s)}{\partial \zeta} q_\pi(s, a), \quad (8)$$

where $d_\pi^\gamma(s) = \sum_{s_0} d(s_0) \sum_{t=0}^{\infty} \gamma^t \sum_a P_\pi(S_t = s | S_0 = s_0)$ is the discounted state occupancy measure, and $P_\pi(S_t = s | S_0 = s_0)$ the probability of reaching state s from s_0 in t steps when following policy π . This update has been the basis of many modern algorithms, including the well-known proximal policy optimization (Schulman et al., 2017).

3.2 Hierarchical Reinforcement Learning

Hierarchical reinforcement learning extends the usual RL formalism by explicitly representing and exploiting the modular, compositional structure argued for in Section 2. It does so by factoring the policy π into a library of sub-policies, each governing behaviour over a stretch of time, and a mechanism that determines how they are invoked. This idea has a long history in AI, where such sub-policies are commonly referred to as skills (Fikes and Nilsson, 1971; Thrun and Schwartz, 1994). Several mathematical formalisms make this idea precise, including options, goal-conditioned policies, and others. We mainly adopt the options framework (Sutton et al., 1999b; Precup, 2000) as it provides a widely adopted and principled formalism for expressing this structure. In Section 3.2.3, we expand on how alternative formalisms connect to the option framework.

3.2.1 OPTIONS: A MATHEMATICAL FORMALISM

An HRL agent makes use of a set of options, $\mathcal{O}^2$, which are defined by three components: a *policy*, an *initiation function*, and a *termination function*. These components can be implemented through parameterized functions, such as neural networks, or symbolically through code (e.g., see Section 6 and Section 9.3). In Figure 4, we illustrate the temporal structure exhibited by options while interacting with an environment. More formally,

Definition 1 (Option) An option $o \in \mathcal{O}$ is a triple $o = \langle \mathcal{I}, \pi, \beta \rangle$, where:

- $\pi : \mathcal{S} \times \mathcal{O} \rightarrow \Delta(\mathcal{A})$ is an option policy,³ which selects an action⁴ according to the current state and the current option.⁵ When parameterized by weights θ , we write $\pi_\theta(a | s, o)$.

2. The option set may be either (i) finite or countably infinite (discrete), or (ii) uncountably infinite, which can be equipped with a continuous parameterization (see Definition 4) (da Silva et al., 2012).

3. It can also be referred to as the skill’s policy, the intra-option policy, or the goal-conditioned policy (see Section 3.2.3 for more details).

4. More generally, the option policy may select over both primitive actions and other options, i.e., $\pi : \mathcal{S} \times \mathcal{O} \rightarrow \Delta(\mathcal{A} \cup \mathcal{O})$, enabling more general hierarchical structures. We restrict option policies to primitive actions here, following the standard definition by Sutton et al. (1999b) for simplicity.

5. We slightly abuse the notation here with respect to the symbol representing the policy of a non-hierarchical RL agent. The distinction between the two will be clear through context. A separate notational convention concerns the option argument in these functions: it indexes the components associated with an option, rather than referring to the option triple itself. We retain $\mathcal{O}$ as the argument domain for notational convenience here and introduce explicit option identifiers later in Definition 4.

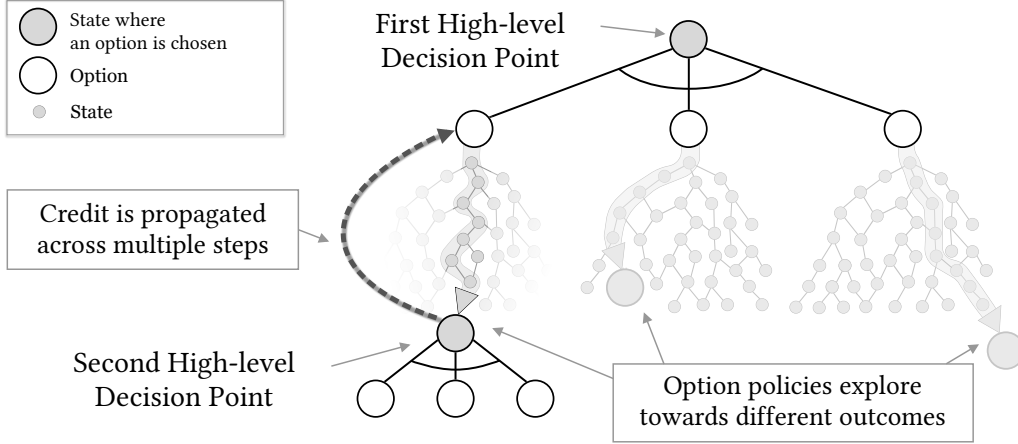


Figure 4: A simplified diagram illustrating the decision process of a hierarchical agent. Large white nodes denote high-level decisions over options, which are assumed to be discrete here for simplicity. Large grey nodes represent the state observed by the agent when a high-level decision is made. Grey trajectories represent state transitions during option execution, where the agent follows the option policy until it terminates. Different options can last for different durations and once an option terminates, the agent makes its next high-level decision.

- $\beta : \mathcal{S} \times \mathcal{O} \rightarrow [0, 1]$ is the termination function, giving the probability that an option o terminates in state s . When parameterized, we write $\beta_\psi(s, o)$.
- $\mathcal{I} : \mathcal{S} \times \mathcal{O} \rightarrow \{0, 1\}$ is the initiation set indicator, specifying whether an option o is permitted to start in state s .⁶

Additionally, to select among the set of options, an HRL agent often uses a *high-level policy* $\mu : \mathcal{S} \rightarrow \Delta(\mathcal{O} \cup \mathcal{A})$, which outputs a probability distribution over the set of options $\mathcal{O}$ and actions $\mathcal{A}$ given a state s . When this probability is parameterized, we use the notation $\mu_\kappa(o|s)$, where κ represents the high-level policy parameters. Similarly to the previous components, in practice, this policy can also be instantiated by other means, e.g., a programmatic policy that directly encodes domain knowledge or follows predefined rules (see Section 9.3).

When planning with options, an agent can make use of an *option model*, comprising a transition model, $P_{\mathcal{O}} : \mathcal{S} \times \mathcal{O} \times \mathcal{S} \times [0, 1] \rightarrow [0, 1]$, which gives the discounted probability that option o , initiated in state s , will terminate in state s' , marginalizing over all possible durations, and a reward model, $R_{\mathcal{O}} : \mathcal{S} \times \mathcal{O} \times [0, 1] \rightarrow \mathbb{R}$, which gives the expected cumulative discounted reward collected during the option’s execution. Both quantities depend on the discount factor γ and enable planning at the level of options without reasoning over individual timesteps (Sutton et al., 1999b).

Not all of the papers covered in this work will explicitly instantiate each of these components. It is common for research in HRL to only highlight the components for which

6. In practical implementations, the binary initiation condition is relaxed into a continuous membership function for differentiable parameterization. In such cases, one may write $\mathcal{I}_\chi(s, o) \in [0, 1]$ to represent the degree to which state s belongs to the initiation set of option o .

a significant contribution is made and to make assumptions about the other components. For instance, the termination function can be assumed to output termination after a fixed number of timesteps. Similarly, the option initiation function is often assumed to allow option initiation across the whole state space. We will discuss this further in Section 3.2.2 and highlight the relevant aspects within the presentation of each method.

Using these terms, we can now define the option value function,

$$q_\pi(s, o) = \sum_a \pi(a \mid s, o) q_u(s, o, a), \quad (9)$$

where $q_u : \mathcal{S} \times \mathcal{O} \times \mathcal{A} \rightarrow \mathbb{R}$ is the value of executing action a in the context of a state-option pair:

$$q_u(s, o, a) = r(s, a) + \gamma \sum_{s'} p(s' \mid s, a) u_\beta(o, s'). \quad (10)$$

The function $u_\beta : \mathcal{O} \times \mathcal{S} \rightarrow \mathbb{R}$ is called the option-value function upon arrival, that is, the value of executing option o upon entering a state s' is given by:

$$u_\beta(o, s') = (1 - \beta(s', o)) q_\pi(s', o) + \beta(s', o) v_\mu(s'). \quad (11)$$

Finally, the function $v_\mu : \mathcal{S} \rightarrow \mathbb{R}$ is defined as the value function over a set of options,

$$v_\mu(s) = \sum_o \mu(o \mid s) q_\pi(s, o). \quad (12)$$

We say an option is *executed* when, upon its selection, the agent takes actions according to its policy $\pi(\cdot \mid s, o)$, terminating at each state s' with probability $\beta(s', o)$. How options are translated into action selection in this way, and when the agent reconsiders its choice of options, is called the *execution model* and we discuss it in detail in Section 7. Equations 9–12 hold under the general *intra-option* setting, in which the high-level policy and option policies are evaluated at every timestep; a notable special case is *call-and-return*, and we discuss that next.

The Semi-Markov Decision Process (SMDP). Building on an MDP, $\langle \mathcal{S}, \mathcal{A}, R, p, \gamma \rangle$, an SMDP allows decisions to persist for a random, variable duration, so that both transitions and rewards are defined over temporally-extended intervals (Bradtke and Duff, 1994).

Notably, when a high-level policy $\mu(o \mid s)$ selects an option and the agent executes it until termination (the *call-and-return* execution model), the induced decision process over options forms an SMDP (Sutton et al., 1999b). In this case, the decision space is the (fixed) option set $\mathcal{O}$, and the transition kernel is given by

$$p(s', \tau \mid s, o), \quad (13)$$

where τ is the (random) duration until termination governed by the option policy π_o and option’s termination function β . The corresponding option-level reward aggregates discounted rewards accrued during execution:

$$r(s, o) = \mathbb{E} \left[\sum_{k=0}^{\tau-1} \gamma^k R_{t+k+1} \mid S_t = s, O_t = o \right]. \quad (14)$$

Because the state at each option-initiation point is a sufficient statistic for future evolution, the SMDP inherits the convergence guarantees of standard dynamic-programming and RL algorithms, now applied at the level of options rather than primitive actions. The practical consequence is that the agent plans over a much shorter effective horizon: a sequence of hundreds of primitive steps may reduce to a handful of option-level decisions, making both learning and planning substantially more tractable.

In what follows, we will primarily adopt the standard MDP formulation rather than the SMDP formalism for two reasons: (i) the SMDP view relies on the call-and-return assumption, which does not hold when options can be interrupted or when the agent combines multiple option policies to select primitive actions;⁷ and (ii) an induced MDP representation can be obtained by marginalizing over the option duration τ , which can be sufficient in practice. More generally, in hierarchical settings with multiple levels, each level makes decisions whose effective durations are determined by lower-level dynamics and termination mechanisms, so these durations need not be modeled explicitly at that level. That is, given an MDP and a set of options, an SMDP can be induced (cf. the result “MDP + options = SMDP” by Sutton et al. (1999b)).

3.2.2 OPTION DISCOVERY

In the preceding sections, we introduced options (Definition 1) and studied prediction and control given a fixed, predefined set of options. In general, however, such a set might not be specified a priori and must instead be *discovered* from data. This gives rise to the problem of *option discovery*,⁸ namely the algorithmic construction of options.

Definition 2 (Option Discovery) Option discovery *is the problem of constructing a set of options $\mathcal{O}$, i.e., specifying the option triple $\langle \mathcal{I}, \pi, \beta \rangle$ for each option $o \in \mathcal{O}$.*

Option discovery often seeks to construct an option set with desired properties, such as supporting effective task performance (Section 4.5), exhibiting behavioural diversity (Section 4.4), enabling faithful trajectory reconstruction (Section 5.1), facilitating exploration, or supporting transfer across tasks (Section 4.9).

Constructing a set of options is combinatorially challenging, as it requires jointly specifying a collection of option triples (Sutton et al., 2023). Consequently, practical approaches adopt additional design principles that make the search space more tractable. We first highlight common and largely orthogonal design principles in the literature, and then introduce an algorithm template for HRL with option discovery in Algorithm 1.

Design Principle 1: One objective per option. Although the option definition itself does not require optimizing any objective, many approaches focus on *purposeful* options, where each option is designed to achieve an *associated objective*, analogous to how an RL agent is driven by a reward function (Littman, 2017). We denote the objective associated with an option o by $\omega_o \in \Omega$, where Ω is a set of admissible objectives. In such cases, the option triple (typically the policy) can be derived or learned from this objective. We refer to options with associated objectives as *subgoal options* (Bagaria, 2025), and a large body

7. These execution models will be discussed in Section 7.

8. Also termed “skill discovery,” encompassing temporal abstraction discovery beyond the option formalism.

of work on option discovery can be understood from this perspective (e.g., McGovern and Barto, 2001; Precup, 2000; Colas et al., 2022; Sutton et al., 2023).

A common and general way to specify such an objective is through an *option reward*.

Definition 3 (Option Reward) *An option reward function is a mapping*

$$r^o: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R},$$

which specifies a reward signal conditioned on an option o . When parameterized, we denote the option reward by r_ν^o with parameters ν .

This signal is distinct from the environment reward $r(s, a)$ because it is internal to an option on the agent side. It is also referred to as “*pseudo-reward*” (Dietterich, 1998) or “*subgoal reward*” (Bagaria, 2025) in the literature. The corresponding option policy $\pi(\cdot \mid s, o)$ is typically obtained by (approximately) maximizing the expected return induced by r^o . However, not all subgoal options are defined through reward maximization: some approaches derive option behaviour from demonstrations (e.g., Team et al., 2024), and others specify it through symbolic methods such as code (see Sections 6 and 9.3). What unifies these approaches is that the option’s behaviour is purposeful, however the objective is specified.

Design Principle 2: Constrained initiation and termination. The search space can be restricted by imposing structure on (a subset of) the option triple $\langle \mathcal{J}, \pi, \beta \rangle$, thereby reducing degrees of freedom in the specification. Examples include fixing the initiation function to constrain admissible states (e.g., Bacon et al. (2017) assume $\forall s, \mathcal{J}(s) = 1$), and restricting the termination function to control when options can end. A common instance of the latter is to augment termination with a maximum step count H_o , introducing a clock that forces termination after a fixed horizon. This can be viewed as allowing β to depend on an internal time index, thus rendering the option non-Markov. An alternative that preserves the Markov property is to use a constant per-timestep termination probability (e.g., $\beta = 1/H_o$), yielding the same expected duration while preserving the Markov property. Both formulations can be interpreted as restricting the class of admissible termination functions.

Design Principle 3: Structured option representation. The option space can also be restricted by representing options with an identifier $z \in \mathcal{Z}$ whose structure, via a shared parametric form (e.g., $\pi(a \mid s, z)$), enables *generalization* across options (e.g., via geometry or semantic meaning), which resembles the idea of function approximation (see Section 3.1). In earlier sections, we have used o to refer to the identifier naming an option when no ambiguity arises. We now make this distinction explicit by separating the notion of an option from how it is denoted.

Definition 4 (Option Identifier) *An option identifier is an element $z \in \mathcal{Z}$ that uniquely denotes an option, with an associated option $o_z = \langle \mathcal{J}_z, \pi_z, \beta_z \rangle$.*

This abstraction isolates the combinatorial object (the option triple) from its parameterization. In the literature, the identifier z is commonly instantiated as follows:

- *Option index and latent variable.* In the classical options framework, options are elements of a finite, discrete set $\mathcal{O}$, and z is simply an index (often denoted as “ i ”)

Algorithm 1 Hierarchical Reinforcement Learning with Option Discovery.

Require: Environment Env , options $\mathcal{O}$, high-level policy μ , data $\mathcal{D}$, option objectives Ω .

```

1: procedure OPTIONDISCOVERY( $\mathcal{O}, \mathcal{D}, \Omega$ )
2:    $\Omega \leftarrow$  OPTIONSPECIFICATION( $\mathcal{O}, \mathcal{D}$ )            $\triangleright$  Define or refine option objectives
3:    $\mathcal{O} \leftarrow$  OPTIONLEARNING( $\mathcal{O}, \mathcal{D}, \Omega$ )          $\triangleright$  Optimize option components
4:   return  $\mathcal{O}, \Omega$ 
5: end procedure

                                      $\triangleright$  Main procedure below

6: for  $t = 0, 1, \dots$  do
7:    $\mathcal{D} \leftarrow$  INTERACT( $Env, \mathcal{O}, \mu, \mathcal{D}$ )            $\triangleright$  Collect experience
8:    $\mathcal{O}, \Omega \leftarrow$  OPTIONDISCOVERY( $\mathcal{O}, \mathcal{D}, \Omega$ )    $\triangleright$  See lines 1–5
9:    $\mu \leftarrow$  POLICYIMPROVEMENT( $\mu, \mathcal{O}, \mathcal{D}$ )          $\triangleright$  Improve the high-level policy
10: end for
    
```

selecting $o \in \mathcal{O}$ (Sutton et al., 1999b). In some methods, z is sometimes treated as a latent variable, sampled from a prior distribution and inferred from interaction via a learned posterior or discriminator (Eysenbach et al., 2019; Kipf et al., 2019). This enables discovering options via variational or mutual-information objectives. In particular, this inference enforces only that z is separable from different behaviour. Thus, in either case, $\mathcal{Z}$ can impose a strong prior by restricting the option space to a fixed, typically small set of behaviours.

- *Continuous representation or embedding.* Some approaches endow z with geometric structure, viewing it as a continuous representation in which nearby identifiers correspond to behaviourally similar options (Kim et al., 2019). This supports interpolation and generalization across options.
- *Goal or subgoal.* In goal-conditioned formulations, z corresponds to a goal or subgoal (often presented as “ $g \in \mathcal{G}$ ”), and options are defined conditionally on z , with the policy and sometimes the termination function derived from that goal specification (Schaul et al., 2015; Andrychowicz et al., 2017). This imposes a semantic interpretation on z . See the discussion on goal-conditioned RL in Section 3.2.3.
- *Program, code, and natural language.* In some approaches, z lies in a space of symbolic descriptions that compositionally specify procedural behaviour, e.g., sequences or trees of primitives (Andreas et al., 2017; Konidaris et al., 2018; Sun et al., 2020). Here, $\mathcal{Z}$ carries compositional (syntactic) structure with procedural semantics, in contrast to goal-based formulations that specify desired outcomes. Additionally, z can also be expressed in natural language (Jiang et al., 2019), which can exhibit compositional structure, albeit less formally than symbolic representations (see Section 6). In such approaches, options are derived from z through an additional decoding mechanism, and z often affords interpretability (Section 2.1).

Bringing the preceding components together, Algorithm 1 outlines a generic HRL template with option discovery, with diverse instantiations detailed in Sections 4–6. It interacts with an environment Env and is initialized with a set of options $\mathcal{O}$, a high-level policy μ , available

data $\mathcal{D}$ (possibly empty), and an initial set of option objectives Ω (possibly empty, described in Section 3.2.2). Each iteration (lines 7–9) operates as follows.

- **INTERACT** (line 7) collects data by executing actions according to the options and the high-level policy, appending experiences to $\mathcal{D}$; in offline settings, this step can be vacuous since the dataset is fixed in advance. This entails specifying how options (and, if applicable, the high-level policy) are used to generate primitive actions, i.e., the execution model (Section 7).
- **OPTIONDISCOVERY** (line 8, implemented in lines 1–4) maps the current set of options $\mathcal{O}$, data $\mathcal{D}$, and option objectives Ω to updated options $\mathcal{O}'$ and objectives Ω' . The specification step determines the objectives, and learning step produces the behaviour that achieves them.
- **OPTIONSPECIFICATION** (line 2) defines or refines the set of objectives, Ω , that options should achieve. In many cases where Design Principle 1 in Section 3.2.2 holds, each option objective determines *what* the associated option should do, for example by specifying an option reward function r^o (Definition 3), or a subgoal g . Methods without this assumption instead often use a single shared objective for all options, in which case this procedure always outputs a fixed objective.
- **OPTIONLEARNING** (line 3) optimizes the option components—policies $\pi(\cdot|s, o)$, termination functions β , and initiation functions J —given those objectives Ω .
- **POLICYIMPROVEMENT** (line 9) updates the high-level policy μ to better select among the current options, if a high-level policy is applicable.

To illustrate how these operations interact concretely, consider the example of a robot that should acquire an option for grasping an object. **OPTIONSPECIFICATION** would identify grasping as a useful subgoal and define a corresponding option reward r^o that provides a positive signal when the object is held. **OPTIONLEARNING** would then train the option policy $\pi(\cdot|s, o)$ to maximize r^o and derive appropriate termination and initiation conditions—perhaps to initiate the option when the object is in the vicinity of the robot, and terminating when it has been successfully grasped. **POLICYIMPROVEMENT** would update μ to learn when selecting the grasping option is useful for the task at hand.

The template is deliberately general; different methods vary in how they instantiate each operation and in design choices. First, $\mathcal{D}$ denotes the data available to the agent and may consist of newly collected online experience, a growing replay buffer, a fixed offline dataset, or any combination thereof; the template does not prescribe how data is obtained. Second, the ordering and frequency of the operations may differ: some methods interleave all four operations at every iteration, while others separate discovery into a distinct phase—calling **OPTIONSPECIFICATION** *before* deploying the resulting options objectives, possibly without further modification. More generally, these correspond to different update ratios, where one specification step may be followed by multiple learning steps. As we will see in Sections 4–6, both paradigms are common in the literature. Third, not all methods explicitly invoke every operation. Some specify options with fixed objectives and only learn the option components (see Algorithm 7 and 11); others expand the option set without ever revising

existing options (see Section 4.1.1); others still do not learn a high-level policy at all, relying instead on programmatic or language-based planners (see Section 9.3) or generalized policy improvement (GPI)-style updates (see Section 7.2.2). What the template makes explicit is that, regardless of the specific method, **HRL with option discovery can be understood as a combination of these operations.**

3.2.3 RELATED TERMINOLOGIES AND FORMALISMS

The previous section uses the language of options to formalize temporal abstractions. As the field of HRL is rich and diverse, some researchers may feel misrepresented by such a formalism. Therefore, we adopt the options framework as main formalism throughout, but use “options”, “skills”, and “goal-conditioned policies” following the terminology used in the original work, and occasionally refer more broadly to temporal abstractions. We believe that, despite differences in terminology, formalism, and representation, the underlying ideas are closely aligned. We now highlight the differences among various related formalisms and illustrate their connections to options.

Skills. The skill terminology has largely been used *informally* in the HRL literature to refer to temporally-extended behaviours. Skills can be formalized using the options framework (e.g., Konidaris and Barto, 2009b), but they can also be formalized using other formalisms such as macro-actions (Fikes and Nilsson, 1971), feudal hierarchies (Dayan and Hinton, 1993), partial policies (Thrun and Schwartz, 1994), MAXQ (Dietterich, 1998), and HAMs (Parr and Russell, 1997). Barto and Mahadevan (2003) compare those different formalisms. Moreover, in recent work, this term is also used to denote an option identifier (Definition 4), such as a latent variable or embedding (see Section 4.4 and Section 5 for details).

Feudal RL. In Feudal RL (Dayan and Hinton, 1993), decision-making is divided across multiple levels of the hierarchy, where higher-level “managers” set subgoals for lower-level “workers” who are rewarded by their managers for achieving these subgoals. Managers operate under information hiding, relying on an abstracted state representation that omits lower-level details handled by the workers. See Section 4.5 for details.

HAMs. Hierarchies of Abstract Machines (Parr and Russell, 1997) constrain the agent’s behaviour through partial programs specified as finite-state machines (with a call hierarchy). The key idea is that some decisions are fixed by the program while others are left as *choice points* where the agent must learn what to do. For example, a HAM might fix that after picking up an object the agent must carry it to a door, but leave the agent to learn *which* object to pick up. This reduces the effective policy space: the product of a HAM and an MDP yields a reduced SMDP whose optimal policy is the best policy consistent with the constraints imposed by the machine (Parr and Russell, 1997). A sequence of actions between consecutive choice points implicitly defines an option, but unlike the options framework, HAMs additionally constrain the order in which temporally-extended behaviours can be invoked through the machine’s control flow. This structure must be specified by the designer.

MAXQ. The MAXQ framework (Dietterich, 1998) decomposes the task into a hierarchy of subtasks organized as a directed acyclic graph. Its key insight is a recursive decomposition of the action-value function: $q(s, i) = v_o(s) + c(s, i)$, where $v_i(s)$ is the value of completing subtask M_i and $c(s, i)$ is the *completion function*—the expected reward after M_i terminates

under the parent policy in the hierarchy. This decomposition enables state abstraction within each subtask, since $v_i(s)$ need only depend on state variables relevant to M_i . In a nutshell, the designer specifies a hierarchy of subtasks, which induces the recursive value-function decomposition. The value functions and policies for the subtasks are then learned within this fixed hierarchy. Once learned, each subtask can be viewed as exhibiting option-like behaviour during execution.

General Value Functions (GVFs). Value-based RL can be viewed as learning a single piece of predictive knowledge. By contrast, *general value functions* (GVFs) $v_\pi^c(s)$ extend this notion by allowing arbitrary *cumulants* $c \in \mathcal{C}$, thereby representing a rich collection of predictions about the agent’s observation stream (Sutton et al., 2011). Formally, a GVF is defined as

$$v_\pi^c(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \left(\prod_{k=0}^{t-1} \gamma(s_k) \right) c(s_t, a_t, s_{t+1}) \mid s_0 = s \right], \quad (15)$$

where the cumulant c specifies the signal of interest and the discount $\gamma : \mathcal{S} \rightarrow [0, 1]$ may depend on the state, enabling termination-like semantics.

This contrasts with procedural knowledge, which specifies how to act, as embodied by options, skills, or behaviours discussed earlier. Procedural knowledge corresponds to policies (e.g., π , or option policies π_o) and their execution semantics (e.g., initiation and termination), whereas predictive knowledge corresponds to value functions or models that estimate consequences of behaviour. GVFs thus occupy the predictive side: they do not directly prescribe behaviour, but instead encode structured predictions that can inform or support procedural components. In this sense, GVFs and options are complementary rather than disjoint abstractions: options specify temporally extended behaviour, while GVFs encode predictions about the consequences of that behaviour. Moreover, prior work has shown that options themselves can be specified or characterized through GVF-based predictive knowledge, including predictions about subgoals, termination events, or expected outcomes (Sutton et al., 2023).

Goal-conditioned RL (GCRL). A goal can be formally defined using a triple: (g, r^g, γ_g) , where $g : \mathcal{S} \rightarrow \mathbb{R}^d$ is a goal vector that can, for example, be used to condition a policy, $\pi(a|s, g)$, or a value function, $v(s|g)$, $r^g : \mathcal{S} \rightarrow \mathbb{R}$, is a goal reward function that maps each state to real-valued number, and, finally, $\gamma_g : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ describes the goal’s continuation function, and hence the timescale for achieving that goal (Kaelbling, 1993; Schaul et al., 2015). Many works in *goal-conditioned* RL (GCRL) consider the state space to be the set of goals an agent should reach (e.g., Andrychowicz et al., 2017), although the framework itself is more general and can admit goal-spaces that are distinct from the environment’s state-space (Schaul et al., 2015).

It is natural to view GCRL through the lens of GVFs. In particular, each goal g specifies a goal reward r^g and continuation function γ_g , thereby inducing a goal-contingent prediction about future outcomes. Thus, GCRL can be understood as learning a collection of related GVFs indexed by goals, where each goal defines a distinct predictive question.

A direct consequence of this view is that one could, in principle, learn a separate approximator for each goal-induced GVF, as in the *Horde* architecture (Sutton et al., 2011). However, since goals typically exhibit structure (as described in Section 3.2.2), this motivates sharing representations across goals. Universal Value Function Approximators (UVFAs)

(Schaul et al., 2015) exploit this structure by representing the family of goal-indexed GVF’s with a single function approximator, e.g., $V_\phi : \mathcal{S} \times \mathcal{G} \rightarrow \mathbb{R}$, alongside a goal-conditioned policy $\pi_\theta : \mathcal{S} \times \mathcal{G} \rightarrow \Delta(\mathcal{A})$ that generalizes behaviour across goals. Although some of the works from the GCRL literature are present in this work (e.g., Section 4.6 and 5.2), we refer the reader to Liu et al. (2022) for an in-depth review.

3.2.4 BEYOND ARCHITECTURAL CHOICES

In the previous section, we noted that options, skills, goal-conditioned policies, and related constructs reflect the same underlying principle, but exhibit nuanced differences in their level of formalization and usage. We now clarify this statement by examining how these constructs manifest across different architectural instantiations.

HRL is often associated with a particular architecture. A prominent example is a *modular hierarchical architecture*, where a high-level policy selects among options, each parameterized separately. Another widely used instantiation is the *goal-conditioned policy*, where a single policy network is conditioned on a goal variable (potentially specified by a foundation model such as an LLM; see Section 6). An HRL system may even consist solely of a single, monolithic neural network, where options or goals are not explicitly represented but implicitly encoded in the parameters (Kobayashi et al., 2025).

Despite these architectural differences, all these instantiations share a common principle: they introduce variables (e.g., options or goals) that parameterize temporally-extended behaviour. This commonality indicates that hierarchy arises from *how behaviour is defined and learned*, rather than how the agent is structured. Thus, we argue that **HRL is fundamentally defined through the algorithm, not the agent architecture.**

An HRL algorithm could empower the agent’s exploration by directing behaviour through sub-policies that operate at different timescales. It can also facilitate more effective credit assignment by decomposing a long, continuous stream of experience into meaningful subtasks. Additionally, HRL can better prepare an agent for future challenges by promoting the learning of reusable behaviours, which can be explicitly or implicitly defined. These essentially represent the core benefits of HRL, as outlined earlier in this work in Section 2.1.

4. Discovery from Online Experience

In this section, we present work in option discovery that takes place in the *online* setting: the agent seeks to construct useful options by simply interacting with the environment. This setting has received significant attention because it holds the promise of scalability (Sutton, 2019)—a long-lived agent that can learn new, useful options simply via interaction and can potentially keep increasing its competence in the world, bootstrapping new skills with previously discovered ones (Ring, 1994; Schmidhuber, 2010).

We broadly categorize this literature based on the proxy objectives and principles used for option discovery. For each family of methods, we first describe the core intuition and key methodological patterns. Then, we discuss how each category contributes to the core benefits of HRL (as outlined in Section 2.1). Finally, we discuss some limitations of each category and highlight opportunities for research.

Before presenting the methods in detail, we direct the reader’s attention to Table 1, which provides an overview of all the discovery methods discussed in this work. For each

Table 1: A summary of HRL methods shown in Section 4, 5, and 6 that discover temporally abstract behaviours, **highlighting the main benefits** elaborated in Section 2.1. Each method links to the corresponding section. A single black dot (●) indicates that a class of methods generally contributes to addressing a specific challenge, while a double black dot (●●) signifies that the class of methods is explicitly designed to tackle that challenge.

Categories	Methods	Reference	Focus of Methodologies on Different Benefits		
			Credit Assignment	Exploration	Transferability
Discovery from Online Experience	Bottleneck Discovery	4.1	●	●	●
	Spectral Methods	4.2	●	●●	●
	Skill Chaining	4.3	●●		●
	Empowerment Maximization	4.4	●	●●	●
	Via Environment Reward	4.5	●●		●
	Curriculum Learning	4.6	●	●●	●
	Meta Learning	4.7	●	●	●●
	Intrinsic Motivation	4.8		●●	●
	Directly Optimizing the Benefits of HRL	4.9	●	●	●
Discovery through Offline Datasets	Variational Inference	5.1	●	●	●
	Hindsight Sub-goal Relabeling	5.2	●●		
Discovery with Foundation Models	Embedding Similarity	6.1		●●	●
	Providing Feedback	6.2	●	●●	●
	Reward as Code	6.3			●●
	Directly Modeling the Policy	6.4		●	●●

method, we highlight which benefits have been mostly studied by researchers from the field, where a single black dot (●) indicates that a class of methods generally contributes to addressing a specific challenge, while a double black dot (●●) signifies that the class of methods is explicitly designed to tackle that challenge.

4.1 Bottleneck Discovery

Many challenging problems in RL have *bottlenecks*: small regions of states that an agent must pass through to reach a larger, potentially more interesting region of the state space. For example, in the *Two Rooms* task (Sutton et al., 1999b; Solway et al., 2014), the agent must go through a doorway state to access the goal in the other room. Another example is

Algorithm 2 Bottleneck Discovery: Instantiation of Algorithm 1.

```

1: procedure OPTIONSPECIFICATION( $\mathcal{O}, \mathcal{D}$ )
2:    $B \leftarrow \text{IDENTIFYBOTTLENECKS}(\mathcal{D})$   $\triangleright$  e.g., diverse density, min-cut, betweenness
3:    $\Omega \leftarrow \emptyset$ 
4:   for  $b \in B$  do
5:      $r_b^o(s) \leftarrow \mathbb{1}(s = b)$   $\triangleright$  Example subgoal reward
6:      $\Omega \leftarrow \Omega \cup \{(b, r_b^o)\}$ 
7:   end for
8:   return  $\Omega$ 
9: end procedure

10: procedure OPTIONLEARNING( $\mathcal{O}, \mathcal{D}, \Omega$ )
11:   for  $\omega = (b, r_b^o) \in \Omega$  do
12:      $\beta_b(s) \leftarrow \mathbb{1}(s = b)$   $\triangleright$  Example termination condition
13:      $\mathcal{J}_b(s) \leftarrow 1$   $\triangleright$  Example initiation condition
14:     Train option policy  $\pi_b(\cdot|s)$  to maximize  $r_b^o$ .
15:      $\mathcal{O} \leftarrow \mathcal{O} \cup \{(\mathcal{J}_b, \pi_b, \beta_b)\}$ 
16:   end for
17:   return  $\mathcal{O}$ 
18: end procedure
    
```

a player in a video game who must pick up a key to unlock a door that leads to other levels. In these examples, the doorway and the key act as bottlenecks—reaching those states grants the agent access to an entirely new region of interesting states.

Algorithm 2 shows how bottleneck discovery instantiates the generic template of Algorithm 1. OPTIONSPECIFICATION analyzes the dataset $\mathcal{D}$ to identify a set of bottleneck states B and, for each bottleneck $b \in B$, defines a subgoal reward r_b^o that fires upon reaching b . The resulting set of option objectives $\Omega = \{(b, r_b^o)\}_{b \in B}$ is passed to OPTIONLEARNING, which trains an option policy π_b to maximize r_b^o for each bottleneck, derives the corresponding termination and initiation conditions, and adds the complete option to the option set $\mathcal{O}$. Bottleneck methods typically specify β_b and $\mathcal{J}_b$ directly based on the subgoal; for other discovery methods (e.g., the option-critic; Section 4.5 or skill chaining; Section 4.3), they are optimized incrementally; but for simplicity, we choose to put them in the same subroutine either way. POLICYIMPROVEMENT then updates the high-level policy μ to select among the available options, and is unchanged from the generic template.

The specific choices of r_b^o, β_b and $\mathcal{J}_b$ in Algorithm 2 are representative defaults. For example, some methods restrict initiation to states within the partition leading to the bottleneck (Menache et al., 2002), and implementations often augment the termination condition with a maximum step count to avoid indefinite execution. What matters most across bottleneck methods is the IDENTIFYBOTTLENECKS subroutine: the remainder of this section describes the major approaches.

Most algorithms for finding bottlenecks begin with a graph-based view of the MDP: states are treated as nodes and an edge exists between two states (s, s') when the agent can

transition from s to s' in a single timestep:

$$G = (\mathcal{S}, E), \quad e_{s,s'} \in E \iff \sum_{a \in \mathcal{A}} p(s'|s, a) > 0. \quad (16)$$

This graph is constructed from transitions stored in the dataset $\mathcal{D}$, which may have been collected by calls to the INTERACT sub-routine or provided in advance. In this graph, bottlenecks have been identified using the following approaches.

DIVERSE DENSITY

An early approach to option discovery used the concept of diverse density, which measures how much more likely a state is to lie on a successful trajectory than an unsuccessful one. McGovern and Barto (2001) formulate bottlenecks as states with high diverse density and propose a simple algorithm to identify them. Concretely, consider that the agent has a set of successful trajectories $\mathcal{T}^+$ (sequences of states leading to a goal state) and a set of unsuccessful trajectories $\mathcal{T}^-$ (sequences that did not reach the goal state). The diverse density score $DD(s)$ captures the probability that a given state s occurs in successful trajectories and does not occur in unsuccessful trajectories:

$$DD(s) = \prod_{\tau \in \mathcal{T}^+} P(s \in \tau) \prod_{\tau \in \mathcal{T}^-} (1 - P(s \in \tau)), \quad (17)$$

where the probability that a state occurs in a trajectory can be computed in tabular domains using visitation counts:

$$P(s \in \tau) = \frac{\text{Number of times } s \text{ appears in } \tau}{|\tau|}. \quad (18)$$

In practice, this product is computed in log-space to avoid numerical underflow. States with a diverse density greater than a threshold are chosen as bottlenecks, and subgoal options are created to reach them. A drawback is that trajectories must be classified as positive or negative depending on whether they were on the path to a goal state. Stolle and Precup (2002) address this shortcoming by defining diverse density over a *family of tasks*: bottleneck states are those that are repeatedly visited while solving many goal-reaching tasks.

GRAPH PARTITIONING

Under the graph view of MDPs, bottlenecks can be interpreted as “accumulation” nodes—states through which many paths coincide. These states tend to separate loosely connected subgraphs that are otherwise densely connected internally. To see why, Menache et al. (2002) cast the problem of going from a start state s to a goal state g as a Max-Flow problem (Ahuja et al., 1993): the agent should maximize the flow of probability along paths originating at s and terminating at g . By the Max-Flow Min-Cut theorem (Ford and Fulkerson, 1962), this is equivalent to finding the lowest-capacity edges whose removal completely separates s from g .

Specifically, the Q-Cut algorithm (Menache et al., 2002) finds bottlenecks by solving the Min-Cut problem. The nodes of the graph are partitioned into two disjoint sets U and

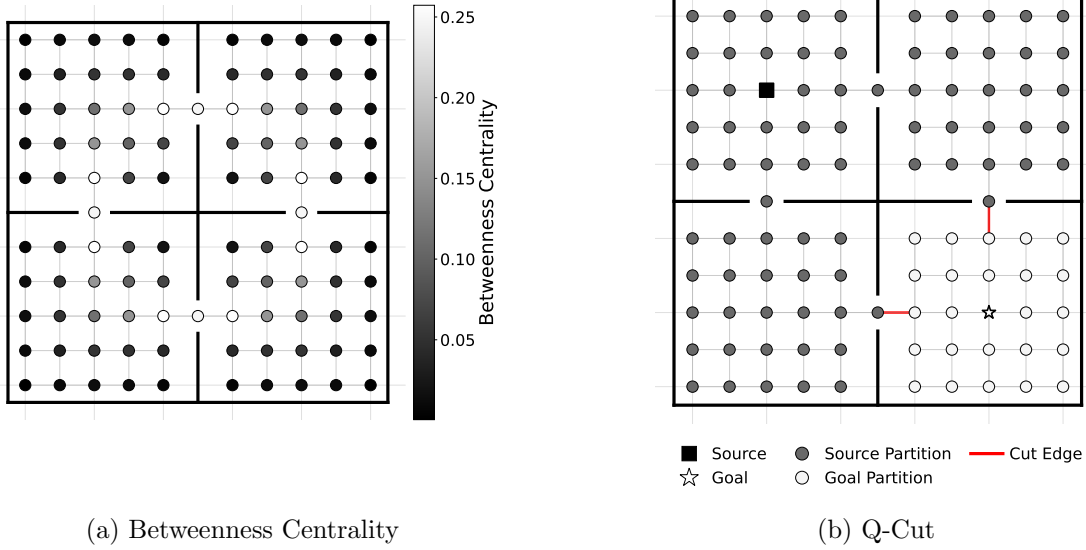


Figure 5: **Bottleneck Discovery in Four Rooms.** Skill discovery using (a) betweenness centrality, a measure of the likelihood that a state lies on the shortest path between any two other states, and (b) Q-cuts, which finds the edge that solves the Min-Cut problem on the transition graph. Both classes of methods attempt to identify bottleneck states and use them as option subgoals.

$\bar{U} = \mathcal{S} \setminus U$, such that the source state belongs to U and the goal state belongs to $\bar{U}$. The cut-value is defined as the total transition probability along edges crossing the partition:

$$\text{Cut}(U, \bar{U}) = \sum_{\substack{(i,j) \in E: \\ i \in U, j \in \bar{U}}} \sum_{a \in \mathcal{A}} p(j|i, a). \quad (19)$$

The min-cut is the partition that minimizes this quantity:

$$U^* = \arg \min_{U \subset \mathcal{S}: s \in U, g \notin U} \text{Cut}(U, \mathcal{S} \setminus U). \quad (20)$$

Although there are exponentially many valid partitions, the Min-Cut problem can be solved in polynomial time (Ford and Fulkerson, 1962). The bottlenecks are defined as the destination nodes of the min-cut edges: $B = \{j \mid (i, j) \in E, i \in U^*, j \notin U^*\}$.

A drawback of Q-Cut is that it requires a global graph over the entire state space, which is not scalable. To address this, L-Cut (Şimşek et al., 2005) constructs local graphs from states visited within a single episode. Rather than searching for individual bottleneck states, Mannor et al. (2004) suggest identifying clusters of states and connecting them using options; this approach has been extended using more sophisticated clustering techniques (Metzen, 2012; Srinivas et al., 2016; Campos et al., 2020; Bacon, 2013). Notably, Evans and Şimşek (2023)’s use of graph modularity (Newman and Girvan, 2004) as the clustering metric allows them to efficiently learn multi-level hierarchies, where each level operates at a different timescale.

GRAPH CENTRALITY

In graph theory, centrality measures the importance of each node within a graph. The search for useful subgoals in an MDP can be viewed as analogous to identifying central nodes. Centrality measures are theoretically well-understood and several efficient algorithms exist for computing them, making them attractive for option discovery. Although many centrality measures exist, Şimşek and Barto (2008) advocate for *betweenness* centrality because of its ability to find bottlenecks in large graphs. Betweenness quantifies how often a node appears on shortest paths between other nodes. Specifically, the betweenness score $b(v)$ for a state v is:

$$b(v) = \sum_{\substack{s, t \in \mathcal{S} \\ s \neq t, s \neq v, t \neq v}} \frac{\sigma_{st}(v)}{\sigma_{st}} w_{st}, \quad (21)$$

where σ_{st} is the number of shortest paths from s to t , $\sigma_{st}(v)$ is the number of those paths passing through v , and w_{st} is a weight assigned to the pair (s, t) . When $w_{st} = 1$ for all pairs, Equation 21 recovers the standard betweenness centrality measure on graphs. To tailor this measure to MDPs, Şimşek and Barto (2008) set w_{st} to the expected reward obtained while going from s to t .

4.1.1 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

It is worth noting how bottleneck methods relate to the phasing distinction raised in Section 3.2.2—that is, whether OPTIONSPECIFICATION is interleaved with data collection or run once as a separate, upfront phase. In principle, bottleneck discovery can be interleaved with interaction, as illustrated by L-Cut (Şimşek et al., 2005), which constructs local graphs from each episode and incrementally updates the set of bottlenecks as the agent explores; this effectively interleaves OPTIONSPECIFICATION, INTERACT, and POLICYIMPROVEMENT. However, many methods instead adopt a modular pipeline, where they first collect data, construct the transition graph, identify bottlenecks, and define subgoal options before deploying them. In such cases, OPTIONSPECIFICATION runs once, which may limit adaptability when new regions of the state space are discovered.

4.1.2 BENEFITS AND OPPORTUNITIES

Exploration. If an agent can reliably reach bottleneck states, it can perform more effective exploration, because states that were once hard to reach become accessible even under a random policy (Şimşek and Barto, 2004 refer to these states as *access states*). For example, picking up a key makes it easy for a video game player to visit previously unseen rooms. When bottleneck discovery is performed incrementally, the agent progressively expands the frontier of its experience in the environment.

Credit Assignment. Methods like those of McGovern and Barto (2001) and Şimşek and Barto (2008) require the agent to solve the problem several times before option discovery can begin; in such cases, exploration is clearly not the main benefit. However, once the agent identifies bottlenecks, it can perform rapid credit assignment for three reasons: (a) value can propagate in large, multi-step jumps from option termination states to option initiation states, (b) value from rewarding events only needs to propagate along trajectories

passing through the bottleneck, greatly reducing the state-action pairs whose values need updating, and (c) in long-horizon problems, the difference in value between actions—the *action-gap* (Farahmand, 2011; Bellemare et al., 2016a)—tends to vanish (Lehnert et al., 2018); partitioning the state space along bottlenecks creates shorter-horizon subtasks with larger action-gaps, easing credit assignment (Lehnert et al., 2018).

Transfer. Bottlenecks are useful for transfer because they are largely task-agnostic—they capture structure in the transition function rather than the reward function, so the same bottlenecks are often useful across a family of tasks. For example, in the *Two Rooms* task, the ability to reliably reach the doorway enables the agent to reach the goal regardless of where it is placed in the second room (McGovern and Barto, 2001). Solway et al. (2014) offer formal support for this view; we discuss their analysis in Section 4.9.

Opportunities for Research.

- *Scalability.* Most methods for finding bottlenecks apply to discrete, explicitly constructed graphs; as a result, they struggle to scale to large, continuous MDPs. In terms of Algorithm 1, the bottleneck is in `OPTIONSPECIFICATION`, which requires building and analyzing the full transition graph. A scalable approach includes spectral methods (Section 4.2), which can compute bottleneck-relevant, continuous properties of the underlying graph without explicitly representing it.
- *Performance guarantees.* With some notable exceptions (e.g., Solway et al., 2014, who relate bottleneck-aligned hierarchies to learning efficiency across tasks), it is generally not well understood how targeting bottlenecks contributes to high-level objectives such as reward maximization or faster planning. Methods outlined in Section 4.9 address this question for option discovery in general, but given the number of bottleneck-specific algorithms, it would be valuable to identify whether there is a high-level objective that is directly optimized (at least approximately) by this family of proxy objectives.

4.2 Spectral Methods

Many option discovery methods are based on the idea of leveraging the state space’s topology, be it to discover options that identify key states that connect different partitions of the environment (Şimşek et al., 2005), that connect states that are far from each other when looking at the diffusion properties of the environment (e.g., Machado and Bowling, 2016; Machado et al., 2017), or that easily allow the agent to traverse the environment in a reusable manner (Liu et al., 2017; Klissarov and Machado, 2023). They are termed spectral methods because they extract information about the state space, such as connectivity or diffusion, through the eigenvectors of a matrix representation of the environment.

The different algorithms in this group leverage the different ways of representing the environment as a matrix and the different types of information one can extract from such matrices. Originally, heavily inspired by results from the graph theory literature, these methods were based on the graph Laplacian and its eigenfunctions,⁹ which can approximate any function on the graph (Chung, 1997). The normalized graph Laplacian, $\mathcal{L}$, for example,

9. Eigenfunctions can be seen as a generalization of eigenvectors to continuous state spaces.

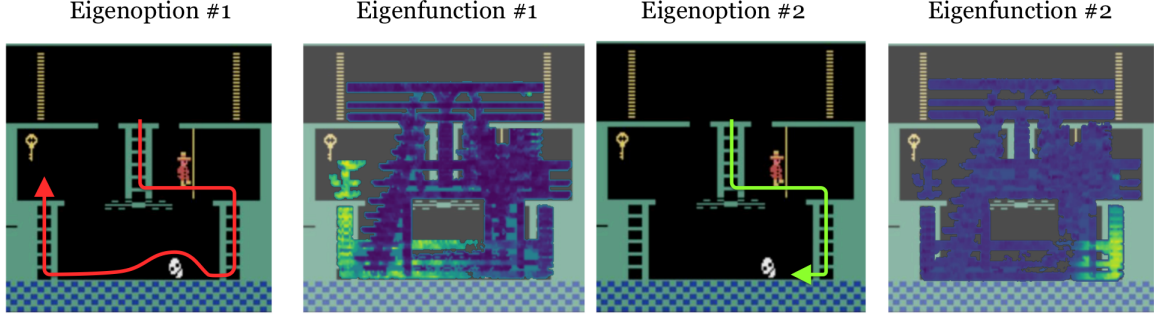


Figure 6: Visualization of the first and second eigenfunctions on Montezuma’s Revenge, an Atari 2600 game, discovered by the algorithm proposed by Klissarov and Machado (2023). The arrows depict what the eigenoption induced by these eigenfunctions could end up being.

is defined as

$$\mathcal{L} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{A})\mathbf{D}^{-1/2}, \quad (22)$$

where $\mathbf{A}$ is the graph’s adjacency matrix obtained by modeling each state in the environment as a node. The adjacency matrix reflects the degree of connectivity between two states. The matrix $\mathbf{D}$ is a diagonal matrix whose entries are the row sums of $\mathbf{A}$. In the reinforcement learning literature, the eigenvectors of the graph Laplacian are also known as proto-value functions (PVFs; Mahadevan, 2005; Mahadevan and Maggioni, 2007).

Importantly, when considering the eigendecomposition, $\mathcal{L}\mathbf{e} = \lambda\mathbf{e}$, the eigenvector of the graph Laplacian associated to the second smallest eigenvalue captures the number of connected components in a graph (Shi and Malik, 2000), allowing one to easily identify bottleneck states (Şimşek et al., 2005), as discussed in Section 4.1. The eigenvectors of the graph Laplacian, in general, capture different time scales of diffusion, which can be used to discover options that promote exploration, such as eigenoptions (Machado et al., 2017, 2018; Machado, 2019), covering options (Jinnai et al., 2019c, 2020), and covering eigenoptions (Machado et al., 2023).

Eigenoptions, for example, are defined such that each option, o_i , is associated with the corresponding eigenvector, $\mathbf{e}_i$, of the graph Laplacian. Their policy is defined as the policy that maximizes the intrinsic reward that incentivizes the agent to navigate alongside the direction pointed by $\mathbf{e}_i$, which, in the linear function approximation (and tabular) case, is formalized as

$$r^{\mathbf{e}_i}(s, s') = \mathbf{e}_i^\top (\phi(s') - \phi(s)), \quad (23)$$

where $\phi(s)$ denotes the feature representation of state s . Originally, an option o_i was defined to terminate in state s if $q_\pi^{\mathbf{e}_i}(s, a) \leq 0$ for all $a \in \mathcal{A}$, where $q_\pi^{\mathbf{e}_i}$ is defined w.r.t. $r^{\mathbf{e}_i}(\cdot, \cdot)$. All other states in the environment were defined to be in the initiation set.

Naturally, explicitly representing an environment through its underlying graph is not scalable. Existing methods leverage approximations of the eigenfunctions of the graph Laplacian that can be obtained through neural networks trained with stochastic gradient descent (Wu et al., 2019; Wang et al., 2021; Gomez et al., 2023). The underlying idea behind these methods is to learn a representation that captures the properties of the approximated eigenvectors such that observations that happen “close in time” are close

in representation space and that different eigenfunctions are indeed orthogonal to each other. The current state-of-the-art method for doing so is called the *augmented Lagrangian Laplacian objective* (ALLO; Gomez et al., 2023). It consists of the following max-min objective for approximating d eigenfunctions:

$$\max_{\boldsymbol{\omega}} \min_{\mathbf{u} \in \mathbb{R}^{d \times |S|}} \sum_{i=1}^d \langle \mathbf{u}_i, \mathcal{L} \mathbf{u}_i \rangle + \sum_{j=1}^d \sum_{k=1}^j \omega_{jk} (\langle \mathbf{u}_j, \llbracket \mathbf{u}_k \rrbracket \rangle - \delta_{jk}) + b \sum_{j=1}^d \sum_{k=1}^j (\langle \mathbf{u}_j, \llbracket \mathbf{u}_k \rrbracket \rangle - \delta_{jk})^2, \quad (24)$$

where $\mathcal{L}$ denotes the graph Laplacian again, $\llbracket \cdot \rrbracket$ the stop gradient operator, δ_{jk} the Kronecker delta, b is a scalar hyperparameter, and $\boldsymbol{\omega} = [\omega_{1,1}, \omega_{2,1}, \omega_{2,2}, \dots, \omega_{d,1}, \dots, \omega_{d,d}] \in \mathbb{R}^{d(d+1)/2}$ is a vector containing all of the dual variables of the objective. Note that the optimal dual variables, $\boldsymbol{\omega}^*$, are proportional to the smallest eigenvalues of $\mathcal{L}$. These approximations have now been used to learn options that are effective in various domains, including continuous control tasks (Jinnai et al., 2020; Shehmar et al., 2026; Chandrasekar and Machado, 2026), 3D navigation tasks, and Atari 2600 games (result shown in Figure 6) (Klissarov and Machado, 2023). An issue these methods had to circumvent was that most of these approximation objectives assume the ability to sample uniformly the entire state space. This is currently addressed by iteratively increasing the region covered by the agent (e.g., Machado et al., 2023); some methods even do so explicitly in the objective they minimize (Erraqabi et al., 2022).

The process to compute the intrinsic reward maximized by the option is slightly different when using neural network estimates of the eigenfunctions of the graph Laplacian. Instead of first computing the eigenvectors, one usually directly estimates the components of the eigenfunction associated with a particular state, s . Formally,

$$r^{f_{e_i}}(s, s') = f_{e_i}(s') - f_{e_i}(s), \quad (25)$$

where we used $f_{e_i}(s)$ to denote the value of i -th eigenfunction of the graph Laplacian associated with state s . In this setting, stochastic option terminations are more common in practice due to the difficulties generalization introduces to accurately estimating action-value functions without interference (e.g., Klissarov and Machado, 2023).

Many other mathematical objects are somewhat equivalent to the eigenvectors of the graph Laplacian and have also been used for option discovery. Slow Feature Analysis (SFA; Wiskott and Sejnowski, 2002; Sprekeler, 2011), for example, are a key component of Continual Curiosity-driven Skill Acquisition (CCSA; Kompella et al., 2017). The eigenvectors of the successor representation (SR; Dayan, 1993) have also been shown to be equivalent to the eigenvectors of the graph Laplacian (Machado et al., 2018).

The equivalence between the eigenvectors of the SR and of the graph Laplacian is particularly important due to the predictive nature of the SR and the ease with which one can learn it incrementally. In fact, the SR now has a quite prominent role in the option literature, being used in the discovery of options for both faster credit assignment (Ramesh et al., 2019) and exploration (Machado et al., 2018; Machado, 2019).

The successor representation is defined as

$$\boldsymbol{\Psi}_{\pi}(s, s') = \mathbb{E}_{\pi, p} \left[\sum_{t=0}^{\infty} \gamma^t \mathbb{1}_{\{S_t = s'\}} \mid S_0 = s \right], \quad (26)$$

Algorithm 3 Spectral Option Discovery: Instantiation of Algorithm 1.

```

1: procedure OPTIONSPECIFICATION( $\mathcal{O}, \mathcal{D}$ )
2:   Update or define the graph representation from  $\mathcal{D}$ .
       $\triangleright$  E.g., eigenvectors or eigenfunctions of the graph Laplacian  $\mathcal{L}$  in Eq. 22
3:   Update pseudo-rewards,  $\Omega = \{r^{e_i}\}$ .
       $\triangleright$  E.g.,  $r^{e_i}(s, s')$  in Eq. 23 or  $r^{f_{e_i}}(s, s')$  in Eq. 25
4:   return  $\Omega$ 
5: end procedure

6: procedure OPTIONLEARNING( $\mathcal{O}, \mathcal{D}, \Omega$ )
7:   For each intrinsic reward  $r^{e_i}$  in  $\Omega$ , learn an option policy  $\pi(\cdot \mid s, o = i)$ .
       $\triangleright$  Example initiation and termination:
       $\triangleright \forall s, \mathbb{J}(s) = 1$ , terminate with probability  $\frac{1}{H_o}$  for each step
8:   return  $\mathcal{O}$ 
9: end procedure

```

where $\mathbb{1}$ denotes the indicator function. The SR was originally introduced through an intuition that is very similar to the one outlined above: one should capture the environment’s dynamics by assigning similar values to temporally close states, thus creating a representation of the underlying structure. It can also be estimated with temporal-difference learning methods (Sutton, 1988), which, as we mentioned above, allows us to learn it incrementally:

$$\Psi(S_t, j) \leftarrow \Psi(S_t, j) + \eta \left(\mathbb{1}_{\{S_t=j\}} + \gamma \Psi(S_{t+1}, j) - \Psi(S_t, j) \right), \quad (27)$$

where we used $\Psi(\cdot, \cdot)$ to denote a sample-based approximation of Ψ_π .

Importantly, beyond the discovery methods mentioned above; as a representation, which was its original purpose, the SR can also be used to combine options without additional learning (Barreto et al., 2019), and recent results in neuroscience and cognitive sciences suggest the SR can model activations in the hippocampus (Stachenfeld et al., 2017; de Cothi and Barry, 2020; Gomez et al., 2025) and explain some human behaviour (Momennejad et al., 2017). These results have led Machado et al. (2023) to propose that the successor representation should be seen as the “natural substrate for the discovery and use of temporal abstractions” in RL.

In terms of scalability, again, there have been many proposals on how to scale the SR to function approximation settings ranging from specific neural network architectures (Kulkarni et al., 2016; Machado et al., 2018; Chua et al., 2024) to ideas such as successor features (Barreto et al., 2017) and successor measures (Touati and Ollivier, 2021; Farebrother et al., 2023). Successor features, for example, can be seen as a projection of the SR onto the space realizable by the representation, ϕ . In matrix form, if we use $\Phi \in \mathbb{R}^{|\mathcal{S}| \times d}$ to denote the matrix encoding the d -dimensional feature representation of each state, successor features are defined as $\Psi_\pi = \sum_{t=0}^{\infty} (\gamma \mathbf{P}_\pi)^t \Phi = (I - \gamma \mathbf{P}_\pi)^{-1} \Phi$.

4.2.1 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

Algorithm 3 provides a general instantiation of Algorithm 1 for spectral methods. The algorithm has two key steps. First, it constructs or updates a graph using the experiences from the dataset $\mathcal{D}$, computes or updates graph representation (e.g., Laplacian eigenvectors), and converts each component into a pseudo-reward that measures progress along that direction in the state space. Second, for each such pseudo-reward, it learns a policy that maximizes it, yielding a set of option policies, each corresponding to a distinct spectral mode of the environment’s structure. For initiation and termination, we show an example from Klissarov and Machado (2023), which corresponds to the discussion in Design Principle 2 of Section 3.2.2.

4.2.2 BENEFITS AND OPPORTUNITIES

Exploration. The eigenoptions line of work (Machado et al., 2017) has popularized the use of temporal abstraction for exploration, alongside studies that highlighted its benefits, including Brunskill and Li (2014) and Fruit and Lazaric (2017). Eigenoptions can significantly decrease the diffusion time¹⁰ in an environment, and this afforded exploration can lead to faster learning. Machado et al. (2018) further extends previous work to the function approximation case by estimating the successor representation and then performing a singular value decomposition on it. Jinnai et al. (2019c) introduce covering options, arguing that rather than constructing an option for every eigenvector of the graph Laplacian, a single option constructed based on the second eigenvector is sufficient. This is because that single option minimizes the cover time of the underlying MDP, which loosely refers to how long it takes for a random high-level policy to visit all states. Leveraging direct approximations of the eigenfunctions of the graph Laplacian, Jinnai et al. (2020) extended covering options to the function approximation case, and Klissarov and Machado (2023) did the same for covering eigenoptions (Machado et al., 2023), demonstrating strong exploration properties in a variety of reinforcement learning problems. More recently, Theophilou and Şimşek (2025) have shown how to augment the Laplacian representation with state-visitation counts, obtaining options that lead to even better exploration in the tabular case, while also being able to better deal with asymmetric environments.

Transferability. Options are often thought to be important in continual (lifelong) learning settings (See also Section 9.2) where skills can be reused in an ever-changing world. The benefit of Laplacian-based options in such settings has been demonstrated both in simpler tabular problems in which the goal location changes regularly (Liu et al., 2017) and in more complex, high-dimensional settings in which not only the goal location would change but also the topology of the environment (Klissarov and Machado, 2023).

Opportunities for Research.

- *Improving Representations.* Machado et al. (2023) have proposed the perspective that spectral methods consist of a phase in which a representation is first learned (e.g., PVFs, SR), followed by a phase in which options are then derived from such a representation. This process can even be done in a cycle, which Machado et al.

10. The diffusion time encodes the expected number of “decisions” required to navigate between two states randomly chosen in an environment (Machado et al., 2017).

(2023) called Representation-driven Option Discovery (ROD) cycle. Thus, better representation learning methods are an exciting research frontier for this line of work in which the learned representation informs the option discovery process. This can be investigated from the SR perspective (e.g., Touati and Ollivier, 2021; Carvalho et al., 2023; Farebrother et al., 2023), or from the perspective of directly estimating the spectral decomposition of the SR (e.g., Pfau et al., 2018; Wang et al., 2021, 2022; Gomez et al., 2023), including non-symmetric settings (Wang et al., 2023b).

- *Planning.* Another promising line of work involves further exploring the recent success of Laplacian-based methods in planning and credit assignment in general, as these options are often used in a reward-agnostic way (e.g., Sutton et al., 2023). Validating these results beyond the tabular case and extending existing results to partially-observable settings are also intriguing lines of work.
- *Reward-Aware Representations.* The representations discussed in this section rely on the topology of the environment without considering the underlying reward function. There is an interesting question of whether one should define proximity not only in terms of when observations take place but also in terms of the reward associated with them. Interestingly, the linear MDP formalism (Todorov, 2006, 2009b) gives rise to representations akin to the SR, but that are reward-aware. In this context, Tse et al. (2025) has shown that options derived from the eigenvectors of such a reward-aware representation, termed the default representation (Piray and Daw, 2021), exhibit qualitatively different exploratory behaviour when faced with regions of negative reward in the state space. More recently, Tse and Machado (2026) has proposed an objective that allows us to use neural networks to estimate the principal eigenvector of the default representation.

4.3 Sequentially Composable Options

Options are said to be *sequentially executable* when each option terminates in a region where another option can successfully achieve its own subgoal. Sequentially composable options are more useful for high-level planning (Konidaris et al., 2018) and can result in highly robust solutions (Tedrake et al., 2010). While most methods attempt to sequentially compose discovered options *post hoc*, some methods explicitly incorporate sequential composition into the option discovery objective. A prominent family of such methods is that of Skill Chaining (Konidaris and Barto, 2009b; Bagaria and Konidaris, 2020).

In the language of Algorithm 1, skill chaining is distinctive in that `OPTIONSPECIFICATION` depends on the output of `OPTIONLEARNING` from a previous iteration: the initiation set learned for the most recent option becomes the subgoal for the next option in the chain. This creates a tight interleaving between specification and learning that contrasts with bottleneck methods (Section 4.1), where `OPTIONSPECIFICATION` can run as a single preprocessing phase. In skill chaining, each iteration of the outer loop in Algorithm 1 adds at most one new option to the chain, and the loop terminates when the start state is covered by some option’s initiation set.

Figure 7 illustrates the algorithm. Given a target region of states $g \subset \mathcal{S}$ (for example, the task goal, shown as a flag in Figure 7), skill chaining discovers subgoal options that

can be sequenced together so that each option execution roughly brings the agent closer to g . This is done by learning options backward from the goal: first, the agent learns option o_1 with termination condition $\beta_{o_1}(s) = \mathbb{1}(s \in g)$ and subgoal reward $r^{o_1}(s) = \mathbb{1}(s \in g)$. Option learning entails training two functions: (a) the option policy $\pi(\cdot|s, o_1)$, which aims to maximize r^{o_1} , and (b) the initiation function $\mathcal{I}(s, o_1)$, which represents the probability that $\pi(\cdot|s, o_1)$ can reliably reach g from state s . Shortly after, the agent creates another option o_2 whose subgoal is the initiation region of the previous option—this is because the agent can reach the goal with high probability from states inside the first option’s initiation region. This process continues until the start state $s_0 \sim \rho_0$, where ρ_0 denotes the environment’s initial state distribution, is inside the initiation region of some option, at which point the agent can simply execute its learned options in sequence to achieve the goal g . Sequential composability is explicitly enforced by setting the subgoal of each new option o_i to be the states in which the previous option o_{i-1} has a high initiation probability, i.e., $\mathcal{I}(s, o_{i-1})$ is greater than some pre-specified threshold $c \in [0, 1]$.

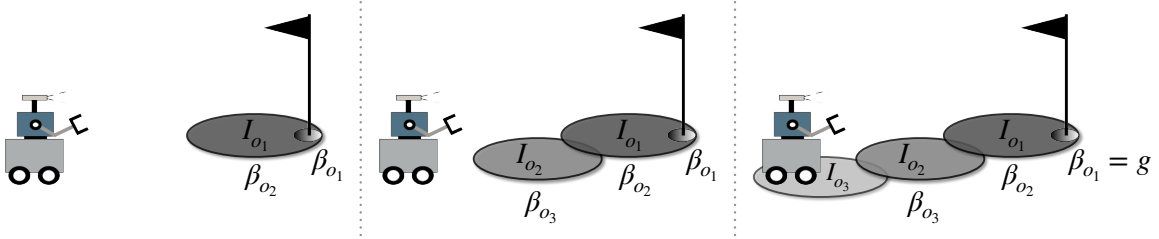


Figure 7: **Sequentially Composable Options.** The skill chaining algorithm incrementally learns options backwards from the goal, such that the subgoal of each option is the initiation region of another option. First the agent finds the states from which it can reliably reach the goal (*left*), then it finds the states from where it can reach the first region (*middle*), and so on, until there is a high probability of success from the environment’s start state (*right*).

In skill chaining, the initiation set of an option has special meaning: it represents the states from which option execution is likely to achieve its subgoal. Learning the initiation function is usually framed as a binary classification problem: states along successful option trajectories (those that achieve the option’s subgoal) are considered as positive examples $\mathbf{s}^+ = \{s_1^+, \dots, s_n^+\}$ and states along unsuccessful trajectories are considered as negative examples $\mathbf{s}^- = \{s_1^-, \dots, s_m^-\}$. Then, a probabilistic classifier (with parameters χ) is fit on these training examples using the binary cross-entropy loss. Now, when a new state s is encountered during learning, $\mathcal{I}(s, o)$ represents the probability that the agent can reach option o ’s subgoal in a single execution of $\pi(\cdot|s, o)$. While this classification approach is simple to implement, some of its drawbacks include: (a) the classifier struggles to adapt to changing option policies, and (b) the agent must wait until the end of option execution to update its initiation function. To address this issue, Bagaria et al. (2023) frame the initiation function as a general value function (Sutton et al., 2011): the agent uses each experience tuple (s, a, r, s') to update its prediction of whether an option execution will achieve its subgoal. Letting $g_o(s)$ denote the option’s subgoal function—a binary function indicating whether state s satisfies option o ’s subgoal—this is done using the following

temporal difference (TD) error and stochastic gradient descent update rule:

$$\delta_J(s, o) = g_o(s') + (1 - g_o(s')) \mathcal{J}_\chi(s', o) - \mathcal{J}_\chi(s, o), \quad (28)$$

$$\chi = \chi - \alpha \delta_J \nabla_\chi \mathcal{J}_\chi(s, o), \quad (29)$$

where $\alpha \in \mathbb{R}^+$ is a step size parameter and χ are the initiation function parameters. However, at a given state s , an option’s initiation probability $\mathcal{J}_\chi(s, o)$ can be low either because the option policy is unlikely to successfully reach its subgoal from state s or because the agent does not have enough data to confidently estimate $\mathcal{J}_\chi(s, o)$. As a result, the skill chaining agent additionally estimates its uncertainty $\mathcal{U}(s, o)$ about its initiation function’s predictions: when deciding whether an option is executable from a state s , it is optimistic with respect to that uncertainty, but when targeting another option’s initiation region, it is pessimistic with respect to it (Bagaria et al., 2021b).

4.3.1 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

Algorithm 4 presents skill chaining as an instantiation of Algorithm 1. INTERACT is defined explicitly because the high-level policy μ is modulated by each option’s initiation probability, so that options are only selected in states where they are likely to succeed. OPTIONLEARNING updates all existing options in $\mathcal{O}$: for each option, it sets the termination condition from the subgoal function g_o , trains the option policy against r^o , and learns the initiation function via temporal difference updates whose cumulant is the subgoal function g_o —this is how the learned initiation function remains grounded in the option’s subgoal.¹¹ When OPTIONSPECIFICATION returns a non-empty Ω , OPTIONLEARNING also creates the corresponding new option and adds it to $\mathcal{O}$. Unlike bottleneck methods where $\mathcal{J}$ is fixed by definition, here the initiation function is genuinely *learned*. POLICYIMPROVEMENT updates μ using extrinsic reward; since this is standard, it is omitted from Algorithm 4. Finally, OPTIONSPECIFICATION checks whether the current chain covers the start state; if not, it returns a new option objective whose subgoal function is derived from the initiation set of the last option in the chain. Note that this decision depends on the initiation functions learned by OPTIONLEARNING in previous iterations, creating a feedback loop between the two subroutines across iterations of Algorithm 1.

4.3.2 BENEFITS AND OPPORTUNITIES

Planning. Each option execution drives the agent to a small, predictable region of the state space. Since those states are constructed to be inside the initiation region of another option, they can be sequentially composed. In practice, each option’s initiation and termination region is parameterized using probabilistic classifiers, so there is a *probability* that two options can be executed in sequence, which eventually permits computation of the probabilistic feasibility of entire plans (Konidaris et al., 2018). Bagaria et al. (2023) used graph-search to find recursively optimal solutions and Bagaria et al. (2021b) provided a dynamic programming algorithm to approximate hierarchically optimal ways of planning with subgoal options.

11. In continuous state spaces, the hard threshold in the subgoal function $g_{o'}(s) = \mathbb{1}(\mathcal{J}(s, o_k) > c)$ is replaced by a soft sigmoid $\sigma(\mathcal{J}(s, o_k) - c)$; we use the indicator for clarity.

Algorithm 4 Skill Chaining: Instantiation of Algorithm 1.

Require: Target region $g \subset \mathcal{S}$, option horizon H_o , initiation thresholds $c_1, c_2, c \in [0, 1]$.

 1: Initialize $\mathcal{O} \leftarrow \{o_1\}$ with $g_{o_1}(s) \leftarrow \mathbb{1}(s \in g)$, $\mathcal{D} \leftarrow \emptyset$.

 2: **procedure** INTERACT($E, \mathcal{O}, \mu, \mathcal{D}$)

 3: Sample option $o \in \mathcal{O}$ from:

$$\frac{\mu(o|s)\mathcal{J}^+(s, o)}{\sum_{o' \in \mathcal{O}} \mu(o'|s)\mathcal{J}^+(s, o')},$$

 where $\mathcal{J}^+(s, o) = \text{clip}(\mathcal{J}(s, o) + \mathcal{U}(s, o), 0, 1)$.

 4: **while** option o does not terminate **do**

 5: Execute $a \sim \pi(\cdot|s, o)$, observe r, s' ; append to $\mathcal{D}$.

 6: **end while**

 7: **return** $\mathcal{D}$

 8: **end procedure**

 9: **procedure** OPTIONSPECIFICATION($\mathcal{O}, \mathcal{D}$)

 10: **if** $\mathbb{E}_{s_0 \sim \rho_0}[\mathcal{J}(s_0, o)] < c_1$ & $\mathbb{E}_{s_0 \sim \rho_0}[\mathcal{U}(s_0, o)] < c_2, \forall o \in \mathcal{O}$ **then**

 11: Let o_k be the last option added to the chain.

 12: $g_{o'}(s) \leftarrow \mathbb{1}(\mathcal{J}(s, o_k) > c)$

▷ New subgoal function

 13: $r^{o'}(s) \leftarrow g_{o'}(s)$

▷ New option reward

 14: **return** $\Omega = \{(g_{o'}, r^{o'})\}$

 15: **else**

 16: **return** $\Omega = \emptyset$

 17: **end if**

 18: **end procedure**

 19: **procedure** OPTIONLEARNING($\mathcal{O}, \mathcal{D}, \Omega$)

 20: **if** $\Omega \neq \emptyset$ **then**

 21: Create new option o' from Ω ; $\mathcal{O} \leftarrow \mathcal{O} \cup \{o'\}$.

 22: **end if**

 23: For all $o \in \mathcal{O}$, set $\beta_o(s) \leftarrow g_o(s) + (1 - g_o(s)) \cdot \frac{1}{H_o}$. ▷ Example termination condition

 24: Sample transitions (s, a, r, s', o) from $\mathcal{D}$.

 25: Update option policy $\pi(\cdot|s, o)$ to maximize r^o .

 26: Update initiation via TD using g_o : $\delta \mathcal{J}(s, o) = g_o(s') + (1 - g_o(s'))\mathcal{J}(s', o) - \mathcal{J}(s, o)$.

 27: Update uncertainty estimate $\mathcal{U}(s, o)$. ▷ E.g., using count-based methods

 28: **return** $\mathcal{O}$

 29: **end procedure**

Credit Assignment. Skill chaining has demonstrated more sample-efficient credit assignment in goal-reaching tasks than non-hierarchical RL, which can be attributed to the following reasons. (1) *Jumpy transitions*: Skill chaining methods usually use the entire T -step option transition $(s_t, o, \sum r_{t:t+T}, s_{t+T})$ to update the high-level policy $\mu(o|s)$. Much like

n -step returns and $TD(\lambda)$ in non-hierarchical RL, this has the effect of rapidly propagating credit among state-action pairs. (2) *Focused next-state distribution*: not only does each option execute for multiple timesteps, but it also guides the agent to states that are closer to the goal. In other words, options in the skill chain modify the agent’s state distribution to make states closer to the goal more likely. Since these states are usually the ones with non-zero values, bootstrapping-based value learning (e.g., TD) progresses more rapidly.

Exploration. Since skill discovery proceeds backward from the goal, the algorithm requires either an exploration policy or a set of demonstration trajectories (Konidaris et al., 2010; Kang and Oh, 2022) that achieve the task goal. This advocates for a view of skill chaining as producing options that are good for exploitation, which can be combined with options that are good for exploration. Deep skill graphs (DSG) (Bagaria et al., 2021a, 2025) overcome this limitation: the agent finds intrinsically motivating states and learns skill chains that connect them to each other; the resulting chains form a graph abstraction of the environment, which is useful for planning. Furthermore, the graph building process has a *Voronoi bias* (LaValle, 1998; Lindemann and LaValle, 2004), meaning that it tends to grow towards parts of the state space where the agent has the least experience.

Opportunities for Research.

- *Goal-reaching options.* To learn the initiation set of each option in the chain, its subgoal must be described using a binary function: either the subgoal is achieved in the current state, or it is not. Such a subgoal description is not universal, as it cannot be naturally used to describe continuing tasks like maximizing velocity or repeating periodic motions. If the initiation cumulant (Bagaria et al., 2023) can be formulated for general reward functions, then skill chaining can be applied to non-goal-reaching tasks as well.
- *Controlling all state variables at the same time.* If we think of states being composed of different state variables, which is a property known as *factoredness* (Boutilier et al., 2000), then skill chaining drives the value of all variables to a certain range of values. In more complex environments, it may be unnecessary, or even impossible, to control all state variables at the same time. Future work could create a version of skill chaining that leverages the factoredness of the state space and only controls a subset of all the factors at any given time.

Additional connections to control theory and motion planning. Lozano-Perez et al. (1984), Mason (1985), and Burridge et al. (1999) popularized the view of policies as *funnels*: these policies drive a large set of ordinary states to a small set of desired states. Policies can be sequentially composed to reach some target set of states by placing the end (narrow part) of each funnel inside the beginning (broad part) of some other funnel. Tedrake et al. (2010); Ames and Konidaris (2019) provided a way to compute these initiation regions for complex, dynamical systems using convex optimization and built robust controllers for fixed-wing UAVs (Tedrake et al., 2010). Meanwhile, Konidaris and Barto (2009b) applied this idea to model-free RL; the broad part of the funnel corresponds to an option’s initiation set, while the narrow part corresponds to the option’s subgoal. Bagaria and Konidaris (2020) then upgraded the skill chaining algorithm with deep learning so that it could be applied

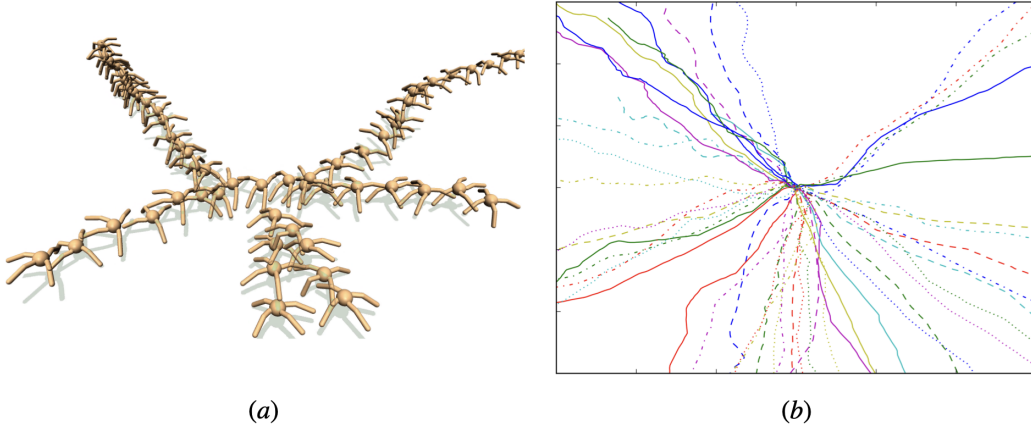


Figure 8: Empowerment-based skill-discovery methods learn skills that generate trajectories that are maximally different from one another, with the constraint that, having observed a trajectory, it should be clear which skill generated it. (a) Trajectories generated by 6 distinct skills in MUJoCo ANT; (b) (x, y) location of the center of mass of the ANT plotted after executing skills learned by the DADS algorithm. Figure from Sharma et al. (2020b), used with permission.

to higher-dimensional systems. Variants of deep skill chaining have been used in robotic surgery (Huang et al., 2023), manipulation (Lee et al., 2021; Vats et al., 2023), multi-agent RL (Xie et al., 2022), and task and motion planning (Mishra et al., 2023).

4.4 Empowerment Maximization

Empowerment-based methods discover diverse skills by maximizing an agent’s control over its environment. At its core, empowerment quantifies how much influence an agent has over its future observations—an agent is more empowered when it can reliably cause a wider variety of outcomes (Klyubin et al., 2005; Salge et al., 2014). For example, having access to a car empowers you to reach many different locations; learning to swim empowers you to survive in water, and so on. Empowerment can also be seen as a way to maximize social influence in multi-agent settings (Jaques et al., 2019), or to seek agreement between future states and the agent’s internal representations (Hafner et al., 2020).

Formally, empowerment is defined as the mutual information between an agent’s actions and its future states. Mutual information $I(X; Y) = \sum_{x,y} p(x, y) \log \frac{p(x,y)}{p(x)p(y)}$ measures how much information one random variable provides about another, equaling zero when the variables are independent and increasing as they become more statistically dependent.

Now, consider an agent that executes a sequence of n actions $\mathbf{a} = (a_t, a_{t+1}, \dots, a_{t+n-1})$ starting from state s_t , resulting in state s_{t+n} . The n -step empowerment at state s_t is:

$$\mathcal{E}_n(s_t) = \max_{p(\mathbf{a})} I(\mathbf{a}; s_{t+n} | s_t), \quad (30)$$

where $p(\mathbf{a})$ is the probability distribution over action sequences that the agent can choose. This captures the maximum amount of information that action sequences can provide about future states, optimized over all possible action distributions $p(\mathbf{a})$. Expanding the mutual

information reveals its intuitive meaning:

$$I(\mathbf{a}; s_{t+n}|s_t) = \mathcal{H}(s_{t+n}|s_t) - \mathcal{H}(s_{t+n}|s_t, \mathbf{a}), \quad (31)$$

where the first term represents uncertainty about future states given only the current state, while the second represents remaining uncertainty after choosing actions. Empowerment measures how much this uncertainty can be reduced through deliberate action choice.

As the mutual information is intractable, Mohamed and Rezende (2015) propose to estimate it through variational inference. Specifically, the authors estimate $p(\mathbf{a}|s_{t+n})$ using the variational approximation $q_\phi(\mathbf{a}|s_{t+n})$ and leverage the non-negativity of the KL divergence to obtain:¹²

$$I(\mathbf{a}; s_{t+n}|s_t) = \mathcal{H}(\mathbf{a}|s_t) - \mathcal{H}(\mathbf{a}|s_t, s_{t+n}) \quad (32)$$

$$= \mathcal{H}(\mathbf{a}|s_t) + \mathbb{E}[\log p(\mathbf{a}|s_t, s_{t+n})] \quad (33)$$

$$\geq \mathcal{H}(\mathbf{a}|s_t) + \mathbb{E}[\log q_\phi(\mathbf{a}|s_t, s_{t+n})] \quad (\text{Variational Bound}). \quad (34)$$

This variational bound (Equation 34) provides a practical way to compute empowerment in high-dimensional continuous spaces using neural networks (with parameters ϕ). However, this objective finds open-loop action sequences $\mathbf{a}$, and we want to discover *skills* (closed-loop policies). Gregor et al. (2017) addressed this concern by introducing Variational Intrinsic Control (VIC), which replaces fixed action sequences with parameterized skills $\pi_\theta(a|s, z)$ conditioned on skill variables z . In the language of options, z is the option identifier (see Definition 4), and each z specifies a distinct option whose policy is $\pi_\theta(\cdot|s, z)$. VIC maximizes the mutual information between skills and final states reached from skill execution:

$$J_{\text{VIC}} = I(z; s_{t+n}|s_t), \quad (35)$$

where s_t is the initial state and s_{t+n} is the final state after executing skill z for n timesteps. We can expand this as:

$$I(z; s_{t+n}|s_t) = \mathcal{H}(z|s_t) - \mathcal{H}(z|s_{t+n}, s_t). \quad (36)$$

Similar to Mohamed and Rezende (2015), VIC uses the variational lower bound:

$$I(z; s_{t+n}|s_t) \geq \mathcal{H}(z|s_t) + \mathbb{E}[\log q_\phi(z|s_{t+n}, s_t)]. \quad (37)$$

This variational lower bound can be optimized by training two neural networks: a policy $\pi_\theta(a|s, z)$ that executes skills, and a discriminator $q_\phi(z|s_{t+n}, s_t)$ that predicts which skill was used based on the final state.

Eysenbach et al. (2019) simplified VIC’s approach in their method *Diversity is All You Need* (DIAYN). While VIC maximizes mutual information between skills and final states, DIAYN instead focuses on making skills distinguishable from the states they visit throughout execution—note the shift from trajectory endpoints to all visited states. DIAYN builds on maximum entropy RL, which augments the standard RL objective with an entropy bonus $\mathcal{H}(A|S)$ to encourage exploration. DIAYN learns skills by maximizing:

$$J_{\text{DIAYN}} = I(s; z) + \mathcal{H}(a|s) - I(a; z|s), \quad (38)$$

12. Note that we are departing from our usual notation to denote p and q using the conventional notation in variational inference.

which has an intuitive interpretation: skills should be distinguishable from the states they visit ($I(s; z)$), actions should be diverse ($\mathcal{H}(a|s)$), but skills should differ primarily through state visitation rather than action choices ($-I(a; z|s)$). This objective can further be simplified by expanding the mutual information in terms of the conditional entropies, and then applying the variational approximation similar to Mohamed and Rezende (2015):

$$J_{\text{DIAYN}} = \left(\mathcal{H}(z) - \mathcal{H}(z|s) \right) + \mathcal{H}(a|s) - \left(\mathcal{H}(a|s) - \mathcal{H}(a|s, z) \right) \quad (39)$$

$$= \mathcal{H}(z) - \mathcal{H}(z|s) + \mathcal{H}(a|s, z) \quad (40)$$

$$= \mathcal{H}(a|s, z) + \mathbb{E}[\log p(z|s)] - \mathbb{E}[\log p(z)] \quad (41)$$

$$\geq \mathcal{H}(a|s, z) + \mathbb{E}[\log q_\phi(z|s)] - \mathbb{E}[\log p(z)]. \quad (42)$$

The final step implies the use of a discriminator $q_\phi(z|s)$ to variationally approximate $p(z|s)$; the result is an intra-skill pseudo reward function for each skill z :

$$r^z(s) = \log q_\phi(z|s) - \log p(z), \quad (43)$$

assuming that the $\mathcal{H}(a|s, z)$ term is maximized using a maximum entropy RL formulation (Haarnoja et al., 2017). This leads to a practical algorithm: sample skill $z \sim p(z)$, execute policy $\pi_\theta(a|s, z)$, train discriminator $q_\phi(z|s)$ to predict skills from states, and update the policy using pseudo-reward in Equation 43.

While DIAYN successfully learns diverse behaviours, Sharma et al. (2020b) observed that it can discover skills with unpredictable effects, making them difficult to sequentially compose downstream. Their method, *Dynamics-Aware Discovery of Skills* (DADS), addresses this by explicitly encouraging predictable skill dynamics while maintaining diversity. Their formulation captures two desirable properties simultaneously: different skills should lead to different future states (diversity), and given the current state and skill, the future state should be predictable. Expanding the mutual information:

$$I(z; s_{t+n}|s_t) = \mathcal{H}(s_{t+n}|s_t) - \mathcal{H}(s_{t+n}|s_t, z) = \mathbb{E} \left[\log \frac{p(s_{t+n}|s_t, z)}{p(s_{t+n}|s_t)} \right]. \quad (44)$$

DADS uses variational approximation with two learned models: a skill dynamics model $q_\psi(s_{t+n}|s_t, z)$ and a marginal dynamics model $q_\xi(s_{t+n}|s_t)$. The skill reward function becomes

$$r^z(s_t, s_{t+n}) = \log q_\psi(s_{t+n}|s_t, z) - \log q_\xi(s_{t+n}|s_t), \quad (45)$$

encouraging skills that make state transitions more predictable than the marginal dynamics. Resulting skills have focused effects, and examples of learned skills are visualized in Figure 8. These methods share the key insight that useful skills should be distinguishable from their effects on the environment—whether through final states (VIC), visited states (DIAYN), or state transitions (DADS), each approach learns discriminators that identify which skill was executed from observations. This creates an intrinsic reward signal driving the discovery of diverse, meaningful behaviours without external supervision.

In terms of option components, most empowerment methods use universal initiation ($\mathcal{J}(s) = 1$ for all skills) and fixed-horizon termination: each skill executes for exactly n steps, after which a new skill is sampled. The skill variable z is sampled once at initiation and

Algorithm 5 Empowerment-Based Skill Discovery: Instantiation of Algorithm 1.

Require: Skill distribution $p(z)$ (e.g., uniform), skill horizon H_o .

- 1: Initialize policy π_θ , discriminator q_ϕ , dataset $\mathcal{D} \leftarrow \emptyset$.
 - 2: **procedure** INTERACT($E, \mathcal{O}, \mu, \mathcal{D}$)
 - 3: Sample skill $z \sim \mu$ where $\mu = p(z)$.
 - 4: Execute $\pi_\theta(\cdot|s, z)$ for H_o steps; append $(s_t, a_t, r_t, s_{t+1}, z)$ tuples to $\mathcal{D}$.
 - 5: **return** $\mathcal{D}$
 - 6: **end procedure**
 - 7: **procedure** OPTIONSPECIFICATION($\mathcal{O}, \mathcal{D}$)
 - 8: Update q_ϕ to predict z from labeled transitions (s, a, r, s', z) sampled from $\mathcal{D}$.
 - 9: **return** $\Omega = q_\phi$ $\triangleright$ Implicitly defines r^z for all z
 - 10: **end procedure**
 - 11: **procedure** OPTIONLEARNING($\mathcal{O}, \mathcal{D}, \Omega$)
 - 12: Compute r^z from $\Omega = q_\phi$ using Eq. 43.
 - 13: Update $\pi_\theta(\cdot|s, z)$ to maximize r^z using transitions sampled from $\mathcal{D}$.
 - 14: $\triangleright \pi_\theta$ implicitly updates all option policies in $\mathcal{O}$
 - 15: All options have $\mathcal{J}(s, o) = 1, \forall s \in \mathcal{S}$.
 - 16: All options terminate after n steps or $\beta(s, o) = 1/H_o$.
 - 17: **return** $\mathcal{O}$
 - 18: **end procedure**
-

held fixed throughout execution, so the conditioned policy $\pi_\theta(a|s, z)$ implements a single closed-loop option for each value of z .

Several extensions address limitations of the basic DIAYN framework. One line of work improves the discriminator: VALOR (Achiam et al., 2018) replaces DIAYN’s per-timestep decomposition $\log q_\phi(z|\tau) = \sum_i \log q_\phi(z|s_{t+i})$ with an LSTM architecture that captures temporal dependencies across the full trajectory, while DISDAIN (Strouse et al., 2022) augments the discriminator with a novelty bonus to counteract its pessimism in unfamiliar states. A second line improves the optimization of the mutual information objective itself; for example, CIC (Laskin et al., 2022) uses contrastive learning for tighter estimation. A third line revisits *what* skills should be distinguishable by; for example, Relative VIC (Baumli et al., 2021) rewards skills for causing characteristic *changes* in state rather than visiting specific states, encouraging transferable behaviours that are defined by their effects rather than their specific endpoints.

4.4.1 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

Algorithm 5 highlights the shared structure across empowerment methods. INTERACT samples a skill $z \sim p(z)$ and stores labeled transitions (s, a, r, s', z) in $\mathcal{D}$, so that subsequent subroutines can identify which skill generated each transition. As in previous sections, the primary variation lies in how OPTIONSPECIFICATION defines option objectives. Here, rather than identifying discrete subgoals, OPTIONSPECIFICATION trains a discriminator q_ϕ

to predict which skill z was executed from observed transitions; the trained discriminator is itself the option objective ($\Omega = q_\phi$), from which the option reward r^z can be derived for any skill on demand (Equations 37–43). A major component that varies across VIC, DIAYN, and DADS is what the discriminator conditions on: final states, visited states, or state transitions, respectively. The option reward is not an indicator function over states but a learned pseudo-reward that evolves during training—a form of non-stationarity shared with other methods such as skill chaining (Section 4.3) and spectral methods (Section 4.2), though arising here from the co-training of q_ϕ with the policy. OPTIONLEARNING trains a single policy $\pi_\theta(a|s, z)$ that implements all options simultaneously, parameterized by z —this contrasts with methods like bottleneck discovery where each option typically has a separate policy. Since updating π_θ simultaneously updates the behaviour of every option indexed by z , the returned policy implicitly represents the entire option set $\mathcal{O}$. Both the discriminator and the policy are updated using labeled transitions sampled from the shared dataset $\mathcal{D}$, which grows as the agent interacts with the environment.

The phasing of operations is also distinctive. OPTIONSPECIFICATION and OPTIONLEARNING are tightly interleaved, with both the discriminator and the policy updated at every iteration. POLICYIMPROVEMENT, by contrast, is often deferred to a separate downstream phase in which a high-level policy μ selects among the discovered skills for a specific task. Some methods partially bridge this gap—for example, DADS learns a dynamics model during skill discovery that can be used for planning at test time (Sharma et al., 2020b)—but a separation between skill discovery and skill use is common in this family.

4.4.2 BENEFITS AND OPPORTUNITIES

Exploration. Empowerment serves as an intrinsic motivation that encourages agents to seek states where they have the most control over future outcomes (Klyubin et al., 2005). Several recent empowerment methods seek to address the exploration problem in RL by explicitly optimizing for skill diversity (Eysenbach et al., 2019) and state-space coverage (Campos et al., 2020); by learning a diverse set of skills, agents not only explore effectively in a single-task setting (Massari et al., 2021), but also adapt quickly to new tasks, reducing sample complexity (Sharma et al., 2020b; Baumli et al., 2021; Hansen et al., 2020). One difficulty in obtaining better exploration with empowerment-based methods is that the learned skills tend to be localized, i.e., only cover a small area. Lipschitz-constrained Skill Discovery (LSD) (Park et al., 2022) replaces mutual information estimation with a Lipschitz-constrained objective, ensuring that learned skills correspond to large, meaningful state transitions rather than minor variations. Exploration is further enhanced by paying attention to the parts of the environment that are within the agent’s control (Park et al., 2023)—this is done by using a controllability-aware distance function that assigns higher values to harder-to-control state transitions, leading to more complex skill acquisition, such as object manipulation, without direct external supervision.

Credit Assignment. While the primary motivation of empowerment-driven techniques is that of exploration, recent methods seek to improve the process of learning policies over discovered skills. For example, Leibfried et al. (2019) derive Bellman operators that combine empowerment- and reward-maximization; Sharma et al. (2020b,a) advocate for learning option models during the skill-discovery phase so that the learned options can

be composed via a planner at test-time. In fact, the simplification of empowerment-based objectives to state-reaching (Pitis et al., 2020) can be viewed as a compromise between learning more expressive skills and creating stationary objectives that ease online policy learning for utilizing discovered skills.

Transfer. Most empowerment-based skill-discovery algorithms learn skills that are distinguishable via specific states encountered in sampled trajectories (for example, the last state of the trajectory). This approach leads to skills that are tied to specific states encountered during skill-learning; in other words, learned skills do not transfer to unseen, related parts of the state space. Relative VIC is a promising approach for learning transferable skills because it rewards skill policies for causing characteristic *changes* in state, rather than targeting specific states themselves. Some algorithms use successor features to enable transfer (Zahavy et al., 2021), but more research is needed on learning skills that simultaneously maximize empowerment and enable reuse across different portions of the state space.

Opportunities for Research.

- *Optimization challenges.* Despite significant progress in online mutual information estimation, empowerment remains challenging to estimate and optimize (Achiam et al., 2018). To address this, Park et al. (2024c) introduce a Wasserstein variant of the mutual information objective, where the KL divergence in MI is replaced with the Wasserstein distance. Finding such ways to ease the estimation and optimization of the empowerment objective is a key area of current research.
- *Connections to causal learning.* Gopnik (2024) hypothesizes that if an agent learns an accurate causal model of the world, it will necessarily increase its empowerment, and, conversely, increasing empowerment will lead to a more accurate (albeit implicit) causal model of the world (Salge et al., 2014). This could enable model-based planning for complex, long-horizon problems, fully unleashing the power of HRL.
- *Possible signatures in human learning.* There is mounting evidence in developmental cognitive science that the drive to learn causal models of the world is behind many of the exploratory capabilities of children (Gopnik and Wellman, 2012). For example, Rovee-Collier and Gekoski (1979) show that infants as young as 3 months old vary their actions to observe their causal effects on their environment; Du et al. (2023b) show that children playing some video games can be thought of as maximizing their empowerment. Gopnik (2024) hypothesizes that empowerment maximization in RL could become the new dominant paradigm (after Bayesian approaches that struggle to scale to large hypothesis spaces) for explaining exploration in humans and other animals.

Connections to goal-based exploration. The empowerment framework connects naturally to goal-conditioned RL, and understanding this connection clarifies the role of the variational distribution in shaping skill discovery. When the variational distribution in Equation 37 is Gaussian and fixed, empowerment objectives reduce to goal-based exploration in RL (Choi et al., 2021), by which we mean methods that propose random target states and use a goal-conditioned policy (Schaul et al., 2015) to reach them, for example, in hindsight experience replay (Andrychowicz et al., 2017) and Go-Explore (Ecoffet et al., 2020). In fact, it is possible to think of goal-based exploration and variational empowerment as lying

on a spectrum: the more expressive the variational distribution, the more powerful, albeit non-stationary, the associated representation learning problem (Choi et al., 2021). Furthermore, Warde-Farley et al. (2019) advocate for taking a mutual information maximization approach to goal-conditioned reward functions, Pitis et al. (2020) argue that empowerment maximization is roughly equivalent to maximizing the size of the set of goals that can be achieved by the agent’s policy, and Levy et al. (2023) find that goal-conditioning can make the empowerment objective significantly easier to compute and optimize. These findings further blur the lines between goal-based exploration in RL and empowerment maximization.

4.5 Via Environment Rewards

Much of the work on learning options has focused on discovering intrinsic reward functions, which are then used to learn option policies. There are, however, two important lines of work that instead aim to learn behaviour directly through the rewards given by the environment.

FEUDAL METHODS

The first set of approaches builds on *feudal reinforcement learning* (Dayan and Hinton, 1993). In this framework, the agent is decomposed hierarchically into managers and workers: managers set subgoals for workers to achieve, and workers use non-hierarchical RL to achieve those subgoals. In this way, goal-setting is decoupled from goal-achievement; each level in the hierarchy communicates to the level below it what must be achieved, but does not specify how to do so. The manager maximizes the reward coming from the environment to define the goals that the worker should achieve. Feudal RL was extended to deep RL through FeUdal Networks (FuNs) (Vezhnevets et al., 2017). FuN learns a two-level hierarchy in which the higher-level manager outputs a goal vector g_t at time t that specifies the direction in which the lower-level worker should modify the agent’s current state. Specifically, a linear transformation, ϕ , then maps the last c goals outputted by the manager into an embedding vector w_t ,

$$w_t = \phi\left(\sum_{i=t-c}^t g_i\right). \quad (46)$$

The worker’s policy is then defined through this embedding vector and a matrix of learnable parameters U_t , that is $\pi^{\text{worker}} = \text{SoftMax}(U_t w_t)$.

The worker policy is trained through the standard policy gradient update rule, where the rewards are defined as,

$$r^{\text{worker}}(s_t, g_{t-i}, s_{t-i}) = \frac{1}{c} \sum_{i=1}^c d_{\cos}(s_t - s_{t-i}, g_{t-i}), \quad (47)$$

where $d_{\cos}(x, y)$ is the cosine similarity measure between vector x and y . The manager policy is learned with the task reward function; however, the authors propose the following update rule, which they term directional policy gradient:

$$\nabla \mu(g_t | s_t) = \nabla d_{\cos}(s_{t+c} - s_t, g_t) (Q^{\text{manager}}(s_t, g_t) - V^{\text{manager}}(s_t)). \quad (48)$$

This update closely puts an emphasis on the direction in which a goal vector points to, and whether that direction was achieved by transitioning from s_t to s_{t+c} .

Algorithm 6 Feudal Methods: Instantiation of Algorithm 1.

```
1: procedure OPTIONSPECIFICATION( $\mathcal{O}, \mathcal{D}$ )
2:   Collect additional goals  $G$  from  $\mathcal{D}$ .
3:   for all  $g \in G$  do
4:     Define the option reward  $r^{\text{worker}}(\cdot; g)$ . ▷ E.g., Eq. 47 and 49.
5:      $\Omega \leftarrow \Omega \cup \{r^{\text{worker}}(\cdot; g)\}$ 
6:   end for
7:   return  $\Omega$ 
8: end procedure

9: procedure OPTIONLEARNING( $\mathcal{O}, \mathcal{D}, \Omega$ )
10:  Update worker policy  $\pi_\theta(a \mid s, g)$  to maximize  $r^{\text{worker}}$ .
11:  Construct options  $\mathcal{O}$ :
     $\pi_o(a \mid s) = \pi_\theta(a \mid s, g), \quad \forall s, \mathcal{J}(s) = 1$ 
    ▷ Goal-induced termination, or terminate after fixed  $H_o$  timesteps
12:  return  $\mathcal{O}$ 
13: end procedure
```

As with most option discovery methods, the high-level policy is trained at the same time as the low-level policy; as the low-level policy changes during learning, data from a high-level action taken in the past may not yield the same low-level behaviour in the future (Nachum et al., 2018). This non-stationarity is addressed using relabeling tricks and off-policy learning in the HIRO algorithm (Nachum et al., 2018). In their work, as well as following literature, the worker’s intrinsic reward is defined as,

$$r^{\text{worker}}(s_t, g_t, s_{t+1}) = -\|s_t + g_t - s_{t+1}\|_2. \quad (49)$$

This definition forgoes the explicit use of the cosine similarity; however, it maintains the idea that a goal vector would represent a delta between state transitions. Later, Levy et al. (2019) present the Hierarchical Actor-Critic (HAC) algorithm, which improves upon HIRO by removing the need for dense reward functions by instead using hindsight experience replay (Andrychowicz et al., 2017). In a separate direction, Hafner et al. (2022) instantiate the feudal architecture within a model-based algorithm called Director, which shows strong performance across a wide range of environments. Their approach additionally provides interpretability as the world model can decode goals into images.

OPTION-CRITIC

The second set of approaches is based on the *option-critic* (Bacon et al., 2017). In this work, the authors derive both the intra-option policy gradient theorem as well as the termination gradient theorem, which provide the update rules for learning option policies and termination functions, respectively. The intra-option policy gradient theorem leads to the following update,

$$\frac{\partial q_\pi(s, o)}{\partial \theta} = \sum_{s, o} d_{\pi, \mu, \beta}^\gamma(s, o) \sum_a \frac{\partial \pi_\theta(a \mid s, o)}{\partial \theta} q_u(s, o, a), \quad (50)$$

Algorithm 7 Option-Critic: Instantiation of Algorithm 1.

```

1: procedure OPTIONSPECIFICATION( $\mathcal{O}, \mathcal{D}$ )
2:   Define the shared objective  $\Omega$  as the environment reward  $r$ .
3:   return  $\Omega$  ▷ No option-specific objectives
4: end procedure

5: procedure OPTIONLEARNING( $\mathcal{O}, \mathcal{D}, \Omega$ )
6:   Update intra-option policies  $\pi_\theta(a \mid s, o)$  using Eq. 50.
7:   Update termination functions  $\beta_\psi(s, o)$  using Eq. 51.
8:   Construct options  $\mathcal{O} = \{(\mathcal{I}, \pi_\theta, \beta_\psi)\}$  with  $\forall s, \mathcal{I}(s) = 1$ .
9:   return  $\mathcal{O}$ 
10: end procedure

```

where, $d_{\pi, \mu, \beta}^\gamma(s, o) = \sum_t \gamma^t P_{\pi, \mu, \beta}(S_t = s, O_t = o)$, is the γ -discounted occupancy measure over state-option pairs. In the policy gradient theorem (Sutton et al., 1999a), the gradient of the flat policy is weighted by the action-value function, leading to an increased probability for actions whose future discounted return is higher. In the case of the intra-option policy gradient, the quantity modulating the action probabilities is the state-action-option value function; therefore, a strict generalization of the policy gradient theorem. The termination gradient theorem used to learn the termination function is derived from the option value of option o upon arrival in state s (see Equation 11). The update rule takes the following form,

$$\frac{\partial u_\beta(s', o)}{\partial \psi} = \sum_{s, o} d_{\pi, \mu, \beta}^\gamma(s, o) \frac{\partial \beta_\psi(s', o)}{\partial \psi} A(s', o), \quad (51)$$

where $A(s', o) = q_\pi(s', o) - v_\mu(s')$ is the advantage function over options, representing how advantageous it is to be in state s' with option o with respect to the value of state s' averaged over all options. Usually, the advantage function is implemented as a heuristic for reducing the variance of the estimator, but in this case, it comes naturally from the derivation of the theorem. Later, Bacon (2018) unified these different objectives and derivation through the following objective,

$$J_\alpha(\omega) = \sum_{s, o} \alpha(s, o) Q_\omega(s, o) = \mathbb{E}_{\alpha, \omega} \left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) \right], \quad (52)$$

where $\alpha : \text{Dist}(\mathcal{S} \times \mathcal{O})$ is a distribution of an initial state-option pair, and where ω defines all the parameters within the options framework, including the termination, option policies, and high-level policy. The authors show how, by assuming independence between the parameters of these components, the previous update rules can be recovered.

The line of work surrounding the option-critic has received significantly more attention than we can present in detail in this section. Some of the contributions include learning safe policies (Jain et al., 2021), using multiple discount factors (Harutyunyan et al., 2019b), learning option termination in an off-policy manner (Harutyunyan et al., 2019a), extending the theorems to multiple levels of hierarchy (Riemer et al., 2018), theoretical derivations that take parameter sharing between options into consideration (Riemer et al., 2020), and learning initiation functions (Khetarpal et al., 2020b).

4.5.1 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

Algorithm 6 is the common instantiation of Algorithm 1 for feudal methods. Feudal RL (Dayan and Hinton, 1993) instantiates this template by defining a goal space and corresponding option reward in `OPTIONSPECIFICATION`, while `OPTIONLEARNING` trains a worker policy to achieve these goals; the manager operates as a high-level policy that selects goals based on environment reward. FuNs (Vezhnevets et al., 2017) follow the same structure, but parameterize goals as directions in a learned representation space and define r^{worker} via cosine similarity (Equation 47), with the worker policy conditioned on an aggregated goal embedding (Equation 46). HIRO (Nachum et al., 2018) retains this hierarchy with option reward in Equation 49, and improves data efficiency via off-policy updates with correction in `POLICYIMPROVEMENT`. HAC (Levy et al., 2019) adapts the template by replacing dense intrinsic rewards in FuN with sparse, goal-reaching rewards and leveraging hindsight relabeling.

Algorithm 7 instantiates the vanilla option-critic algorithm (Bacon et al., 2017). Different variants adopt distinct instantiations. For example, in line 7, Harutyunyan et al. (2019a) replace the standard advantage-based update with a termination critic that optimizes β via

$$\min \mathcal{H}(S_f \mid o), \quad S_f \sim P_\beta(\cdot \mid o), \quad (53)$$

where S_f denotes the random variable of the termination state of option o , and $H(\cdot)$ is the Shannon entropy. For line 8, as another example, Khetarpal et al. (2020b) replace the hard initiation set with a differentiable interest function $I_\chi(s, o)$, and update it according to

$$\chi \leftarrow \chi + \alpha \beta_\psi(s') \frac{\partial \pi_{I_\chi}(o' \mid s')}{\partial \chi} q(s', o'), \quad (54)$$

where $\pi_{I_\chi}(o \mid s)$ is defined as

$$\pi_{I_\chi}(o \mid s) = \frac{I_\chi(s, o) \mu(o \mid s)}{\sum_{o'} I_\chi(s, o') \mu(o' \mid s)}, \quad (55)$$

so that interest is increased for options that, upon termination, are selected in states where they have high option-value.

4.5.2 BENEFITS AND OPPORTUNITIES

Credit assignment. Vezhnevets et al. (2017) show strong performance of their feudal method on a set of Atari 2600 games from the Arcade Learning Environment (Bellemare et al., 2013) and 3D navigation challenges (Beattie et al., 2016). These domains are long-horizon and require the agent to propagate credit across multiple steps. The authors report significantly better results than a baseline not leveraging such a hierarchy.

Transfer. Bacon et al. (2017) present experiments where the learned options improve the ability to generalize across changes in the Four rooms environment (Sutton et al., 1999b) compared to non-hierarchical RL algorithms. Such changes included modifying the goal location and the agent’s starting location. This benefit is later reinforced by multiple works (Zhang and Whiteson, 2019; Khetarpal et al., 2020b; Kamat and Precup, 2020; Klissarov and Precup, 2021) showcasing the transferability of options learned through the option-critic

method in more complex environments such as locomotion control (Todorov et al., 2012) and 3D navigation (Chevalier-Boisvert et al., 2023). In these transfer experiments, the agent usually first learns to perform a task before some component of the task is changed.

Interpretability. A particular highlight of the option-critic line of work is that interpretability naturally emerges by learning options directly from environmental rewards. For example, Bacon et al. (2017) report experiments where the termination function would highlight bottleneck states, which are often seen as key in learning temporal abstraction (Stolle and Precup, 2002). Findings on interpretability are similarly reported across different domains (Klissarov et al., 2017; Harb et al., 2018; Zhang and Whiteson, 2019).

Opportunities for Research.

- *Avoiding option degeneracy.* An important practical obstacle when learning options through the update rules proposed by the option-critic is that it may lead to degenerate solutions. Options tend to reduce to actions where each of the options’ duration is only one timestep long. Another observed phenomenon is that only one option ends up being executed throughout all episodes. In both cases, the essence of temporal abstraction is lost. Bacon et al. (2017) originally noticed such behaviour and imputed it to the theoretical fact that using primitive actions is enough to solve any MDP. To avoid such undesirable behaviour, the authors add a penalty term c_{delib} to the termination gradient’s advantage function: $A(s', o) + c_{\text{delib}}$. This term essentially discourages the termination to prefer switching, unless the advantage in doing so is greater than the value of c_{delib} . A theoretical derivation was later done to justify the use of such a term, which was coined as the deliberation cost (Harb et al., 2018). This cost is introduced as a hyperparameter, which raises the question of what value we should choose for a specific environment. Discovering more general solutions to option degeneration remains an open area of research.
- *Reliance on the environment reward.* The strength of the methods we presented in this section is that they do not require a human-defined objective for learning the hierarchy. As such, these methods heavily rely on an informative environment reward. For example, in feudal methods, if the high-level policy is poorly trained due a sparse environmental rewards, it might output goals that fail to drive the learning progress of the lower-level policy. To address the exploration challenge, recent methods like HAC-Explore incorporate a novelty-based intrinsic rewards (McClinton et al., 2021) or demonstrations (Gupta et al., 2019) to solve longer-horizon tasks.

4.6 Curriculum and Hindsight Learning

Within a complex environment, there exists a diversity of subgoals that are interesting for an agent. Some of these subgoals might be easily achievable, whereas others would simply be impossible to complete for an agent’s current capabilities. How could, then, such an agent learn to achieve difficult goals? An effective strategy would be to try to achieve a curriculum of goals, where the complexity of each attempted goal increases continuously with the agent’s capabilities. The idea of curriculum learning has a long history in AI that goes beyond the RL setting (Kaplan and Oudeyer, 2003; Schmidhuber, 2004; Bengio et al.,

2009; Schmidhuber, 2011). A central question then becomes, how should one prioritize which goal or task should be attempted at any given time? Taking the HRL perspective, we can rephrase this question as: how should the high-level policy select the next goal? This question can be formalized through the following objective function:

$$\max_{\mu} \mathbb{E}_{g \sim \mu} [\mathbb{E}_{\pi} [r^g]] \quad (56)$$

where the goal-conditioned policy, π' , used in the expectation, $\mathbb{E}_{\pi'}[\cdot]$, depends on the choice of the goal selection distribution μ . Here, μ can be interpreted as a high-level policy over goals. Specifically, π' is obtained by starting with an initial policy π_0 and applying N iterative updates. For each iteration $k = 1, \dots, N$, a goal $g_k \sim \mu$ is sampled, and the policy is updated via $\pi_k = U_{g_k}(\pi_{k-1})$, where the update rule U_{g_k} aims to maximize the reward r^{g_k} associated with goal g_k , i.e. through policy gradient updates. The final policy used in Equation 56 is $\pi' = \pi_N$. The optimization will thus find the distribution μ which, when used to update π , would lead to the best performance as measured across all goals $\mathcal{G}$.

The objective of Equation 56 is also referred to as the global learning progress (LP), and is, as such, intractable. Researchers have thus approximated this objective through local measures of LP_{local} (Stout and Barto, 2010; Baranes and Oudeyer, 2013; Forestier et al., 2017; Colas et al., 2018), which can be defined as,

$$\text{LP}_{\text{local},g} = V_{\pi_t,g} - V_{\pi_{t-i},g}, \quad (57)$$

where $V_{\pi_t,g}$ is the estimate of the performance of the updated policy after t iterations on goal g and $V_{\pi_{t-i}}$ is that of the policy at iteration $t - i$. These values are usually obtained through Monte Carlo estimates by rolling out policies over multiple episodes, thus possibly covering a subset of the possible goals within the goal space.

In such methods, the high-level policy μ is often optimized through multi-arm bandit algorithms rather than through RL. In other words, μ maximizes the following one-step reward: $\max_{\mu} \mathbb{E}[r^{\mu}] = \mathbb{E}[\text{LP}_{\text{local},g}]$, which can then be defined as

$$\mu_t(g) = \frac{\exp(|E_t(g)|/e)}{\sum_{g' \in \mathcal{G}} \exp(|E_t(g')|/e)}, \quad (58)$$

where e is the temperature and E_t is an exponential moving average of the rate of change in performance on goal g ,

$$E_{t+1}(g) = (1 - \alpha)E_t(g) + \alpha \text{LP}_{\text{local},g}. \quad (59)$$

The global learning process objective can be approximated through other means, which we discuss in the following sub-section on the benefits and opportunities.

In addition to covering methods that produce curricula by explicitly generating goals according to a certain distribution, we also discuss *implicit curricula*. In these methods, certain properties of the learning algorithm itself create a curriculum-like effect. A prominent example of an implicit curriculum is hindsight experience replay (HER) (Andrychowicz et al., 2017), which stores experience generated by seeking a certain goal g in a buffer called an experience replay, and relabels such experience with a variety of other goals g' . We present HER in Algorithm 9, where we highlight the operations that differ from a standard

- 1: Initialize goal-conditioned policy π_θ , high-level policy μ , progress estimates $E(g) \leftarrow 0$ for all $g \in \mathcal{G}$, dataset $\mathcal{D} \leftarrow \emptyset$.

3: Collect additional goals G from $\mathcal{D}$.

5: Define the option reward r^g .

▷ r^g examples: distance to a target outcome, success indicator, etc.

6: $\Omega \leftarrow \Omega \cup \{r^g\}$

7: end for

8: **return** Ω ▷ Defines what each goal-conditioned option should achieve

9: end procedure

10: **procedure** OPTIONLEARNING($\mathcal{O}, \mathcal{D}, \Omega$)

11: Update $\pi_\theta(\cdot \mid s, g)$ to maximize r^g using transitions sampled from $\mathcal{D}$.

12: Construct options $\mathcal{O}$ from learned goal-conditioned behaviours.

$$\triangleright \forall s, \mathcal{I}(s) = 1; \beta(s) = \mathbb{1}\{s \text{ satisfies } g\}$$

```

13:   return  $\mathcal{O}$ 

```

14: end procedure

15: **procedure** POLICYIMPROVEMENT($\mu, \mathcal{O}, \mathcal{D}$)

16: Estimate a learning progress score LP_g for goals $g \in \mathcal{G}$ from $\mathcal{D}$. ▷ Eq. 57

17: Update exponential moving average $E(q)$ using Eq. 59.

18: Update goal selection distribution: $\mu(g) \propto \exp(|E(g)|/e)$. ▷ Eq. 58

```

19:   return  $\mu$ 

```

20: end procedure

experience replay. In particular, hindsight relabeling is governed by a goal sampling strategy $\mathbb{S}$, while the high-level policy μ specifies the distribution over original goals used during interaction. In its common form, HER relabels failed trajectories using the achieved final state as the goal, yielding goals that naturally progress from easy to increasingly challenging.

Research on learning from a curriculum of goals has received much more attention than we can cover here, producing a diversity of approximations to the global learning progress (Forestier and Oudeyer, 2016; Matiisen et al., 2017; Kovač et al., 2020; Akakzia et al., 2021). For an in-depth review please see the surveys by Colas et al. (2022) and Portelas et al. (2020).

4.6.1 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

Algorithms 8 and 9 instantiate Algorithm 1 for representative methods of explicit curricula (learning-progress-based) and implicit curricula (HER), respectively. These two instantiations highlight a key distinction in how curricula are realized. In explicit curriculum methods (Algorithm 8), the `POLICYIMPROVEMENT` for the high-level policy μ is the main focus. The high-level goal-selection policy is directly optimized using a learning progress score.

Algorithm 9 Hindsight Experience Replay (HER): Instantiation of Algorithm 1.

Require: Goal relabeling strategy $\mathbb{S}$.

```
1: Initialize goal-conditioned policy  $\pi_\theta$ , dataset  $\mathcal{D} \leftarrow \emptyset$ .

2: procedure INTERACT( $E, \mathcal{O}, \mu, \mathcal{D}$ )
3:   Sample initial state  $s_0$  and goal  $g \sim \mu$ .
4:   Generate trajectory for an episode  $(s_0, a_0, \dots, s_T)$  using  $\pi_\theta(\cdot|s, g)$ .
5:   for  $t = 0$  to  $T - 1$  do
6:     Compute reward  $r_t = r^g(s_t, a_t)$ .
7:     Store  $(s_t, a_t, r_t, s_{t+1}, g)$  in  $\mathcal{D}$ .
8:     Sample (relabel) additional goals  $\mathcal{G}' \sim \mathbb{S}$ .
9:     for all  $g' \in \mathcal{G}'$  do
10:      Compute hindsight reward  $r'_t = r^{g'}(s_t, a_t)$ .
11:      Store  $(s_t, a_t, r'_t, s_{t+1}, g')$  in  $\mathcal{D}$ .
12:    end for
13:  end for
14:  return  $\mathcal{D}$ 
15: end procedure

16: procedure OPTIONSPECIFICATION( $\mathcal{O}, \mathcal{D}$ )
17:   Sample a batch  $B \subseteq \mathcal{D}$ .
18:   for all  $(s_t, a_t, r_t, s_{t+1}, g_t) \in B$  do
19:     Add option reward  $r^{g_t}$  to  $\Omega$ .     $\triangleright$  E.g., success indicator based on achieved goal
20:   end for
21:   return  $\Omega$ 
22: end procedure

23: procedure OPTIONLEARNING( $\mathcal{O}, \mathcal{D}, \Omega$ )
24:   Update  $\pi_\theta(\cdot|s, g)$  using transitions related to  $g$  from  $\mathcal{D}$  to maximize  $r^g \in \Omega$ .
25:   Construct options  $\mathcal{O}$  from learned goal-conditioned behaviours.
26:    $\triangleright \forall s, \mathcal{J}(s) = 1; \beta(s) = \mathbb{1}\{s \text{ satisfies } g\}$ 
27:   return  $\mathcal{O}$ 
28: end procedure
```

In contrast, HER (Algorithm 9) does not optimize μ ; instead, it relies on the relabeling procedure in INTERACT to induce an implicit curriculum via the strategy $\mathbb{S}$, reshaping the training data distribution without modifying the goal selection mechanism. Conceptually, goal relabeling (line 8) and hindsight reward computation (line 10) can be viewed as part of OPTIONSPECIFICATION, as they determine which goals (option objectives) are used to train the goal-conditioned policies (option policies) in OPTIONLEARNING; however, in Algorithm 9 we place them within INTERACT to reflect the implementation. For both algorithms, OPTIONSPECIFICATION typically runs only once and Ω is then fixed.

4.6.2 BENEFITS AND OPPORTUNITIES

Exploration. One of the main benefits of leveraging curriculum learning to achieve goals is that the agent will be continuously pushed to the limits of its capacity. By doing so, it might discover new locations in an environment or learn completely new behaviour from the combination of previously achieved goals. A family of methods approximates the global learning progress, specifically with the intent of seeking intermediate difficulty. Florensa et al. (2018) propose using a Goal Generative Adversarial Network (Goal GAN) to automatically generate a curriculum of tasks for reinforcement learning agents. The method focuses on generating Goals of Intermediate Difficulty (GOID), defined as goals where the agent’s current policy, π , achieves an expected performance v_π within a specific range:

$$\text{GOID}_i := \{g : v_{\min} \leq v_\pi \leq v_{\max}\}. \quad (60)$$

Here, $v_{\min}$ and $v_{\max}$ represent the minimum and maximum desired performance, ensuring goals are neither too easy nor too hard for the current policy π_i . The generator in Goal GAN is trained to output goals within the GOID set, whereas the discriminator is trained to distinguish between goals that are within the set from those that are not. Racanière et al. (2019) introduce a setter-solver paradigm with three criteria represented through values in $[0, 1]$: validity, feasibility, and coverage. Goals are sampled according to the distribution defined by these criteria, allowing for a balanced selection. Their findings highlight that these criteria, along with conditioning on the current version of the environment, are crucial for an effective learning curriculum. Sukhbaatar et al. (2018) instead rely on asymmetric self-play to generate a curriculum of explorative goals in reversible or resettable environments, leading to improved performance on a diverse set of tasks. Campero et al. (2021) train a goal-generating teacher to guide a goal-conditioned student policy by proposing goals that are neither too hard nor too easy, as measured by the number of timesteps to reach the goal,

$$r^\mu = \begin{cases} +a & \text{if } t^+ \geq t^* \\ -b & \text{if } t^+ < t^*, \end{cases} \quad (61)$$

where a, b are hyperparameters that quantify the bonus and penalty, t^π represents the time it took the policy to reach the goal, and t^* is a hyperparameter.

Additionally, HER-based approaches (Andrychowicz et al., 2017; Fang et al., 2018; Yang et al., 2021a) have demonstrated promising results in improving exploration compared to curiosity-driven methods. Curriculum-guided HER (Fang et al., 2019) introduces an explicit curriculum that transitions from curiosity-driven selection early on to goal-proximity focus in later stages, mimicking human-like exploration. Complementing HER, CER (Liu et al., 2019) enhance exploration by introducing a competitive dynamic between two agents learning the same task, where one agent is penalized for revisiting states explored by the other. Many more works show how curriculum learning can help in hard-to-explore environments (Colas et al., 2018; Zhang et al., 2020; Pitis et al., 2020; Colas et al., 2020a).

Transfer. Learning successfully through curricula produces a whole set of behaviours that were previously not seen. Open-Ended Learning Team et al. (2021) leverage population-based training to quantify progress on goal completion within large, open-ended, and procedurally-generated environments and tasks. Through a continuum of task difficulty, the authors show that the resulting goal-conditioned agent can generalize zero-shot to new situations.

Opportunities for Research.

- *Refining the measure of progress.* One of the main challenges in deriving a curriculum of goals is accurately measuring how difficult a chosen goal is for a learning algorithm at a certain point in time. Different heuristics can work well for certain environments, for example, the number of timesteps required for reaching a goal (Campero et al., 2021), or might involve a combination of heuristics (Open-Ended Learning Team et al., 2021). However, such formalizations might not be generally applicable. Ideas from unsupervised environment design, where the environment evolves as well as the agent’s parameters, could be particularly promising (Dennis et al., 2020; Jiang et al., 2021; Parker-Holder et al., 2022; Samvelyan et al., 2023). Another important desideratum is that a curriculum should continuously increase the difficulty of the goals, but should also generate interesting goals. Finding a formalization that would encode a general measure of interestingness and difficulty is still an open question. However, for many tasks of interest, such as tasks where human prior knowledge would be relevant, leveraging foundational models offers a particularly promising way to define such metrics for curriculum learning, as covered in Section 6. One notable work is that of Zhang et al. (2024), which investigates whether an LLM’s common sense can be a good measure of interestingness in open-ended environments.

4.7 Meta Learning

RL algorithms, such as Q-learning, learn policies; meta-RL algorithms, in contrast, aim to learn the RL algorithm itself, or parts of it, to subsequently learn a policy. This creates a bi-level optimization: the algorithm for learning the RL algorithm itself is called the *outer-loop*, while the learned algorithm (which learns a policy) is called the *inner-loop* (Schmidhuber, 1987; Thrun and Pratt, 1998; Beck et al., 2025). The appeal of meta-RL approaches is that if the environment demands certain properties from the RL agent (for example, transferability), then such properties will automatically be learned from data, without the explicit need for careful human ingenuity and design in every part of the training process (Silver et al., 2021).

Typically, a meta-RL algorithm consists of an inner and an outer loop. Within each of these loops, a set of parameters is being maximized. Concretely, let ω_{out} represent the parameters learned by the outer loop, and ω_{in} the parameters learned by the inner loop. These parameters in practice represent a particular subset of the option parameters presented in Section 3. For example, Veeriah et al. (2021) optimize the parameters of the option policies and the high-level policy in the inner loop, while the outer loop updates the termination functions and option reward functions (subgoals). In contrast, Frans et al. (2018) can be viewed as placing the high-level policy in the inner loop, where it is adapted to solve individual tasks, while the low-level policies are treated as shared primitives that are meta-learned across tasks in the outer loop.

META-GRADIENTS

A common instantiation of meta-RL algorithms is through the use of meta-gradients. In the inner loop, the agent updates the inner parameters,

$$\omega'_{\text{in}} \leftarrow \omega_{\text{in}} + \alpha_{\text{in}} \nabla_{\omega_{\text{in}}} J_{\text{in}}(\omega_{\text{in}}), \quad (62)$$

where J_{in} is an arbitrary objective that depends on ω_{in} . To obtain the meta-gradients, we assume that the outer parameters depend on the inner parameters. Data is then collected with the updated inner parameters in order to proceed to the following update,

$$\omega'_{\text{out}} \leftarrow \omega'_{\text{out}} + \alpha_{\text{in}} \nabla_{\omega_{\text{out}}} J_{\text{out}}(\omega'_{\text{in}}(\omega_{\text{out}})), \quad (63)$$

$$\omega'_{\text{out}} \leftarrow \omega'_{\text{out}} + \alpha_{\text{out}} \nabla_{\omega'_{\text{in}}} J_{\text{out}}(\omega'_{\text{in}}(\omega_{\text{out}})) \nabla_{\omega_{\text{out}}} \omega'_{\text{in}}(\omega_{\text{out}}), \quad (64)$$

where $\nabla_{\omega_{\text{out}}} \omega'_{\text{in}}(\omega_{\text{out}})$ encodes how the outer loop parameters affected the updated inner loop parameters. The objectives J_{in} and J_{out} may differ in various ways, such as defining different distributions over tasks.

Veeriah et al. (2021) leverage meta-gradients (Finn et al., 2017; Xu et al., 2018) to learn options in high-dimensional navigation environments. In the inner loop, they update the option policies parameters, θ , and the high-level policy parameters, κ :

$$\theta' \leftarrow \theta + \alpha_{\theta} (G_t - q_{\pi}(s_t, o_t)) \cdot \nabla_{\theta} [\log \pi_{\theta}(a_t | s_t, o_t) - q_{\pi}(s_t, o_t)], \quad (65)$$

$$\kappa' \leftarrow \kappa + \alpha_{\kappa} (G_t^{\mu} - v_{\mu}(s_t)) \cdot \nabla_{\kappa} [\log \mu_{\kappa}(o_t | s_t) - v_{\mu}(s_t)], \quad (66)$$

where G_t is the option policy return (see Section 3.2.1), and G_t^{μ} is an n -step return for the high-level policy defined as $G_t^{\mu} = \sum_{j=1}^n \gamma^j r_{t+j} - \gamma^n c + \gamma^{n+1} V_{\mu}(s_{t+n})$ where c is a switching cost added on option terminations, similar to Harb et al. (2018). The outer loop is instantiated through these updates to the parameters ν of the option reward function r_{ν}^o , and the parameters ψ of the termination function β_{ψ}^o ,

$$\psi \leftarrow \psi + \alpha_{\psi} (G_t^{\mu} - v_{\mu}(s_t)) \nabla_{\psi} \log \pi_{\theta'(\psi, \nu)}(a_t | s_t, o_t), \quad (67)$$

$$\nu \leftarrow \nu + \alpha_{\nu} (G_t^{\mu} - v_{\mu}(s_t)) \nabla_{\nu} \log \pi_{\theta'(\psi, \nu)}(a_t | s_t, o_t). \quad (68)$$

The outer loop updates the option-reward and termination meta-parameters using a new trajectory generated by interacting with the environment using the most recent inner-loop parameters, $\theta'(\psi, \nu)$ and $\kappa'(\psi, \nu)$, which depend on the outer loop parameters. The update in the outer loop assesses the impact of updates to the high-level policy, μ_{κ} , and option policies, π_{θ} , and it may involve a different distribution of tasks than the one used in the inner loop, as is common in meta learning (Finn et al., 2017).

BLACK-BOX META REINFORCEMENT LEARNING

In black-box meta RL (Wang et al., 2016; Duan et al., 2016), an agent interacts with a sequence of different tasks drawn from an arbitrary distribution, $p \in \Delta(\xi)$. Each interaction with a task, or distribution of tasks, is considered a trial, which itself consists of N episodes, represented in Figure 9. During a trial, the agent receives observations, rewards, and termination signals from the environment, where episode termination signals represent the

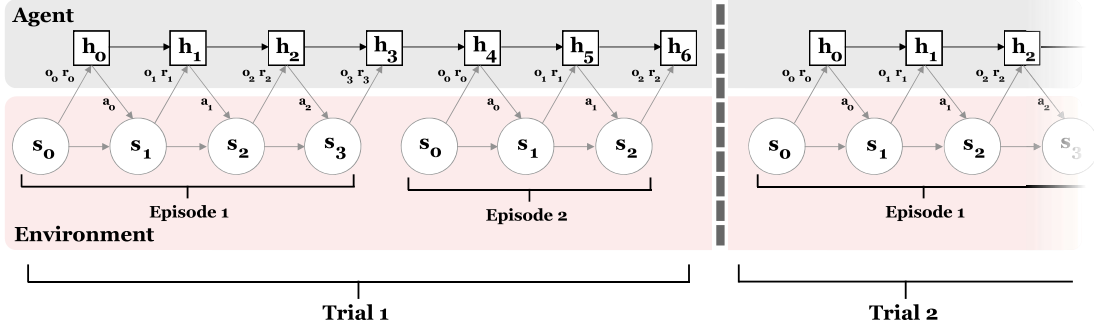


Figure 9: Black-box meta reinforcement learning. Trials consist of multiple episodes during which the hidden state, h_i , of the agent is unrolled. The hidden state is only reset between trials. Figure reproduced from the work of Duan et al. (2016).

episode boundaries. These variables are used to update the agent’s internal memory h , which is typically represented by the hidden state of an RNN (Hochreiter and Schmidhuber, 1997) or the context of a transformer network (Vaswani et al., 2017). Importantly, the agent continuously updates h across episodes within the same trial; the memory is only reset at the end of each trial. The overall goal is to maximize the total reward accumulated over an entire trial,

$$\max_{\pi} \mathbb{E}_{\xi \sim p} \left[\sum_{\text{episode}=1}^N \mathbb{E}_{\pi} \left[\sum_t r^{\xi}(s_t, a_t) \right] \right], \quad (69)$$

where r^{ξ} is the reward associated with task ξ . This objective incentivizes the agent to learn how to adapt its policy based on the experience gathered so far during a trial, effectively forcing it to implicitly learn, through the updates to its policy’s memory h , a reinforcement learning rule capable of efficient adaptation to new tasks. When further conditioning the policy π on the task’s goal g , as done by Bauer et al. (2023), this approach can lead to human-timescale adaptation.

4.7.1 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

We do not provide a generic template related to Algorithm 1 for this section, as different algorithms optimize different components in the inner and outer loops. For example, Frans et al. (2018) optimize a task-specific high-level policy in the inner loop while treating the option policies as shared primitives, and update these shared option policies in the outer loop. Both levels optimize the same task reward, so the method does not instantiate OPTIONSPECIFICATION; the outer loop instantiates OPTIONLEARNING, while the inner loop corresponds to POLICYIMPROVEMENT of the high-level policy. Veeriah et al. (2021) instead learns option policies and the high-level policy using RL on sampled tasks in the inner loop, which corresponds to OPTIONLEARNING, and meta-learns option reward (subgoal) parameters and termination functions via meta-gradients in the outer loop, which corresponds to OPTIONSPECIFICATION. For black-box meta-RL, if the hidden state h is regarded as an option identifier, it can be viewed as an instantiation of Algorithm 1 with an

implicit option space. `OPTIONSPECIFICATION` is trivial, as no explicit objectives are defined beyond the task reward r^ξ . `OPTIONLEARNING` optimizes the option policy $\pi(a \mid s, h)$, while `POLICYIMPROVEMENT` is instead realized through the update rule of h (e.g., via an RNN), which governs how the high-level behaviour evolves over time.

4.7.2 BENEFITS AND OPPORTUNITIES

Transfer. As discussed earlier, a major benefit of learning options is that of reuse: options learned in one part of the state-space could speed up learning in another (Konidaris and Barto, 2007; Taylor and Stone, 2009). Some methods have tried to discover transferable options using meta-learning. For example, `MLSH` (Frans et al., 2018) discovers a set of policies and trains a high-level policy to select among them. The meta-objective trains these components such that the high-level policy can quickly learn to solve new tasks from a distribution by reusing the learned skills, making them reusable across a pre-specified task distribution (Nam et al., 2022; Gupta et al., 2018; Fu et al., 2023). `MODAC` (Veeriah et al., 2021) uses meta-gradients (Xu et al., 2018; Oh et al., 2020) to do the same: an outer loop learns option reward functions and termination functions that an inner loop maximizes using policy gradients. The outer loop of the optimization learns from the reward coming from the environment.

Exploration. Meta-learning approaches have also sought to address the exploration question in non-stationary and multi-task settings. When the agent finds itself in a new environment, how can it systematically leverage prior experience to guide targeted exploration of the new environment? This problem is called *meta-exploration* by Beck et al. (2025). For example, when someone is in a new house and they have to look for utensils, they begin their search from the kitchen; similarly, we would like to create RL agents that can direct their exploration for quick adaptation in new environments (Gupta et al., 2018). This is one of the motivations for the Adaptive Agent (AdA) (Bauer et al., 2023) that uses meta-learning to train a policy capable of human-timescale adaptation in a massive, combinatorial task space (Open-Ended Learning Team et al., 2021). Specifically, they use black-box meta-RL: the policy is implemented as a Transformer-XL model (Dai et al., 2019). This model π_θ takes the history h of interactions within the current episode (past states, actions, rewards) and goal description, g , as input to determine the next action. The adaptation happens implicitly within the recurrent state of the model. The combinatorial complexity of the environments allows for careful selection of tasks that are at the appropriate difficulty given the current agent capabilities, generating an effective meta-learning curriculum. As such, AdA mixes ideas from meta-learning as well as curriculum learning, which we cover in the next section.

Opportunities for Research.

- *Relaxing the multi-task formulation.* Meta-learning approaches have demonstrated abilities of transfer, adaptation, and meta-exploration—abilities that have been challenging to scalably acquire using other techniques. Furthermore, meta-learning via in-context learning (Dong et al., 2022a; Bauer et al., 2023; Raparthy et al., 2023) provides a scalable, and potentially simpler way, to acquire these crucial capabilities. Existing meta-learning approaches rely on a specialized multi-task formulation, with clear task boundaries and episodes. Methods that lift these assumptions will be able

to bring these capabilities to a wider variety of settings. For an in-depth review of meta learning approach, please refer to Beck et al. (2025).

4.8 Intrinsic Motivation

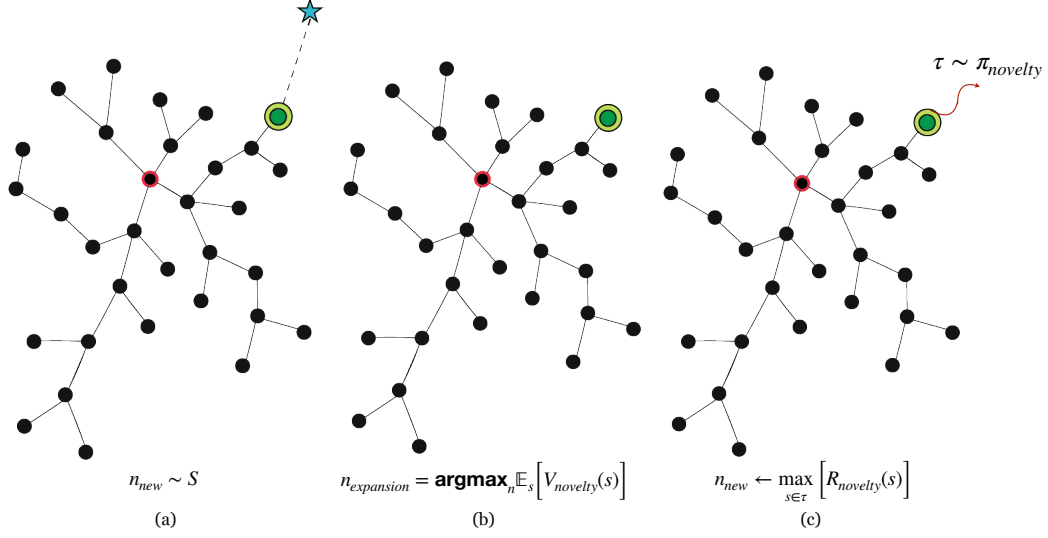


Figure 10: (a) In the original DSG algorithm, a state is sampled uniformly at random (blue star) from the state-space $\mathcal{S}$ and the graph is pulled towards it via its nearest neighbor node (green). (b) The IM-DSG agent uses intrinsic motivation to identify a node to expand using an exploration value function $v_{novelty}$. (c) Once the agent reaches the expansion node, it executes an exploration policy $\pi_{novelty}$, and the most novel state in the resulting trajectory is identified as a target for a new skill.

Intrinsic Motivations (IM) drive actions for their own sake, meaning that they are not in service of achieving an obvious, externally specified goal; instead, they are in service of augmenting knowledge and learning skills whose utility only becomes apparent later on (Oudeyer and Kaplan, 2007; Barto and Şimşek, 2005; Berlyne, 1960; Harlow, 1950). Computationally, this can be formalized using notions of information gain (Bellemare et al., 2016b)—an agent may take actions that result in new information about its environment, even if it requires forsaking extrinsic reward in the short term. IM underpins a developmental approach where an agent learns reusable skills autonomously, preparing it for various future challenges. For example, children, as intrinsically motivated biological agents, develop skills by engaging in activities that yield interesting, memorable outcomes (Gopnik et al., 2009); these skills improve in efficiency with repetition and can be strategically reproduced for specific goals (Barto et al., 2004). Such behaviours are well represented by options, with the intended outcomes encapsulated in the options’ subgoals (Singh et al., 2004). While we have discussed IM-based option discovery approaches like empowerment maximization, spectral methods, and bottleneck discovery, this section explores additional methods not covered by these categories.

NOVELTY-BASED DISCOVERY

One common intrinsic motivation signal is novelty, which decreases with repeated state visitations. For example, upon visiting state s , an agent might receive

$$r^{\text{int}}(s) = N(s)^{-1/2}, \quad (70)$$

where $N(s)$ is the visit count. This count-based bonus is one of several common choices for encouraging exploration of infrequently seen states by making familiar states less rewarding (Auer et al., 2008; Strehl and Littman, 2008; Bellemare et al., 2016b; Lobel et al., 2023). An example of using this for option discovery is provided by the Relative Novelty algorithm (Şimşek and Barto, 2004). Here, a state s is deemed a good subgoal if it leads to experience that is significantly more novel than the experience preceding it. Let $n(s)$ be a novelty score (e.g., inverse visitation count from Equation 70), and let w^+ and w^- be fixed-size windows of future and past states, respectively. The relative novelty at time t is then computed as

$$\text{RN}(s_t) = \frac{1}{|w^+|} \sum_{s \in w^+} n(s) / \left(\frac{1}{|w^-|} \sum_{s \in w^-} n(s) \right). \quad (71)$$

States with high relative novelty are likely to be gateways to unexplored regions (e.g., doorways in Figure 5), and can be automatically selected as subgoals. An option is then created to reach each such subgoal from a broader initiation set, often by learning a dedicated policy using intrinsic reward. In this way, the agent transforms spikes in novelty into reusable skills without knowledge of external task rewards.

A related example is First Return, then Explore (FRTE) (Ecoffet et al., 2020), which formalizes intrinsic motivation using count-based novelty over discretized “cells”, which are pre-specified state abstractions that serve as option subgoals (an example state abstraction is spatially downsampling image-based observations). Every time a new cell is encountered, the agent logs it in an archive. The policy is then conditioned to return to these cells to deepen exploration. FRTE selects target cells based on their inverse visitation count, returning to underexplored frontiers before taking random, exploratory actions. This loop results in a growing archive of reachable states, each effectively defining an option subgoal. This return-then-explore strategy also has a model-based counterpart: Bagaria et al. (2025) extend the Deep Skill Graphs algorithm discussed in Section 4.3 to image-based observation spaces where a meaningful distance metric is not readily apparent. Their algorithm, *Intrinsically Motivated Deep Skill Graphs* (IM-DSG), learns a graph-based model of the world—nodes of the graph represent option subgoal regions (abstract states) and edges represent option policies (abstract actions). Figure 10 illustrates the main steps of the algorithm: first, the agent picks an existing node based on how much that node is expected to contribute to exploration (Figure 10(b)), then the agent plans with its abstract model using dynamic programming to determine the options it needs to execute to reach the sampled node, from where it executes a novelty-seeking exploration policy (Figure 10(c)). States visited by the novelty-seeking policy are candidates for creating a new node in the graph, similar to the Relative Novelty algorithm (Şimşek and Barto, 2004) discussed earlier.

The novelty-based methods above identify *where* in the state space to create option subgoals. A complementary approach is to identify *what aspect of the state* an option should control, using the causal structure of factored MDPs. A factored MDP (Boutilier et al., 2000) assumes the state can be described by a vector of state variables and that the transition dynamics factorize according to a dynamic Bayesian network (DBN). HEXQ (Hengst, 2002) was an early algorithm in this category. HEXQ automatically decomposes a factored MDP into subtasks by detecting *exits*—states where the transition dynamics of a lower-level state variable can no longer be predicted independently of higher-level variables. More formally, if X and Y are state variables ordered by their rate of change (faster-changing variables are lower-level), an exit is identified where the conditional entropy $\mathcal{H}(Y' | X, a)$ increases sharply, indicating that a transition in X cannot be explained without accounting for Y . Each such transition is marked as an exit and treated as a subgoal. HEXQ then learns a hierarchy of options, each option driving the agent toward one of these subgoals—lower-level options correspond to frequently-changing variables, whereas higher-level options handle more slowly-changing aspects. In this way, HEXQ yields an option hierarchy spanning multiple levels of temporal abstraction.

A similar approach is proposed by Jonsson and Barto (2006), who analyze the structure of a learned DBN to extract a causal abstraction hierarchy. For each action, they examine the DBN’s parent-child dependencies: if variable X_i influences the next-state value of X_j , i.e., $X_i \in \text{Parent}(X'_j | a)$ for some action a , then X_i is said to causally affect X_j . These dependencies define a directed graph over state variables, which is decomposed into strongly connected components (SCCs). The SCCs are then topologically ordered to yield levels of abstraction—variables in earlier components are controlled first, while later components are conditioned on them. This is because earlier variables in the topological ordering tend to be those that causally influence, but are not influenced by, variables in later components. While their algorithm assumes access to the transition model, Vigorito and Barto (2008) propose a model-free algorithm that incrementally builds DBNs online through exploration. When new dependencies are detected—e.g., when variable X_i begins to influence X_j —a new option is instantiated to induce that dependency reliably. This learning process can be guided by structure learning techniques (e.g., maximizing Bayesian Information Criterion), or by identifying transitions that lead to salient changes in state abstractions. More recently, Nayyar and Srivastava (2024) cluster states based on the temporal-difference (TD) error:

$$\delta_t = r_t + \gamma v(s_{t+1}) - v(s_t), \quad (72)$$

which serves as a proxy for learning progress. Regions with high variance in δ_t are recursively split, producing a symbolic abstraction over state variables and spawning new options targeted to regions where prediction error remains high. This method resembles HEXQ, but instead of focusing on the frequency of variable changes, it uses the TD-error incurred by candidate state abstractions.

One downside of these factored approaches is that they assume the agent observes factored state variables, which requires significant domain knowledge. Bagaria et al. (2026) address this limitation by developing an agent that learns to identify relevant features directly from image observations. When their agent encounters a particularly novel state,

Algorithm 10 Novelty-Based Option Discovery: Instantiation of Algorithm 1.

```

1: procedure OPTIONSPECIFICATION( $\mathcal{O}, \mathcal{D}$ )
2:   Update novelty scores  $n(s)$  from visitations in  $\mathcal{D}$  ▷ e.g., Equation 70
3:    $B \leftarrow \{s : \text{RN}(s) > c_{\text{RN}} \text{ and } s \text{ is not the subgoal of any } o \in \mathcal{O}\}$ 
▷ e.g., relative novelty spikes (Equation 71)

4:    $\Omega \leftarrow \emptyset$ 
5:   for  $b \in B$  do
6:      $r_b^o(s) \leftarrow \mathbb{1}(s = b)$  ▷ Example subgoal reward
7:      $\Omega \leftarrow \Omega \cup \{(b, r_b^o)\}$ 
8:   end for
9:   return  $\Omega$ 
10: end procedure

11: procedure OPTIONLEARNING( $\mathcal{O}, \mathcal{D}, \Omega$ )
12:   for  $\omega = (b, r_b^o) \in \Omega$  do
13:      $\beta_b(s) \leftarrow \mathbb{1}(s = b); \mathcal{J}_b(s) \leftarrow 1$  ▷ Example termination and initiation
14:      $\mathcal{O} \leftarrow \mathcal{O} \cup \{(b, \pi_b, \beta_b)\}$ 
15:   end for
16:   Train each option policy  $\pi(\cdot|s, o)$ ,  $o \in \mathcal{O}$ , to maximize  $r^o$ .
17:   return  $\mathcal{O}$ 
18: end procedure

```

it uses counterfactual analysis (specifically, they use the DeepSHAP algorithm; Shapley, 1953; Lundberg and Lee, 2017; Shrikumar et al., 2017) to isolate which visual features are responsible for the novelty of that state. Then, the agent learns a classifier that focuses only on these salient features (Singh et al., 2004), ignoring other aspects of the image. This feature-specific classifier serves as an abstract subgoal (Bagaria and Schaul, 2023) for option learning, enabling factored skill discovery without requiring pre-specified state decompositions.

4.8.1 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

As in previous sections, the primary variation lies in how OPTIONSPECIFICATION selects subgoals; these methods do it through internally generated signals such as novelty scores, relative novelty spikes, or structural features like causal exits. Algorithm 10 presents the template for novelty-based discovery, where OPTIONSPECIFICATION thresholds a novelty criterion (here, relative novelty with threshold c_{RN}) to select subgoals. Option construction follows the same defaults as bottleneck discovery (Algorithm 2), but with two differences that reflect the interleaved setting: OPTIONSPECIFICATION only creates options for subgoals that are not already targeted by an existing option, and OPTIONLEARNING continues to train all option policies as new experience arrives. As in the other instantiations, Ω denotes the newly specified objectives; the objectives of existing options persist within $\mathcal{O}$. A distinctive feature of novelty-based methods is that these criteria are often *non-stationary*: novelty scores decrease as the agent accumulates experience, so OPTIONSPECIFICATION must be periodically re-invoked as new data arrives from INTERACT. This tight interleaving between

INTERACT and OPTIONSPECIFICATION resembles the incremental pattern seen in algorithms such as L-Cut (Section 4.1) and skill chaining (Section 4.3). Factored approaches like HEXQ (Hengst, 2002) and the causal abstraction method of Jonsson and Barto (2006), by contrast, solve OPTIONSPECIFICATION as a one-time analysis of the DBN structure, similar in phasing to bottleneck preprocessing. The online variant of Vigorito and Barto (2008), which incrementally builds the DBN through exploration, is closer to the interleaved paradigm.

4.8.2 BENEFITS AND OPPORTUNITIES

Exploration. Intrinsic motivation is perhaps most often used to facilitate exploration, especially when extrinsic rewards are sparse, delayed, or misleading—although early methods were equally motivated by the acquisition of reusable skills (Singh et al., 2004; Barto and Şimşek, 2005). By rewarding novelty, surprise, or learning progress, IM helps the agent identify and prioritize skills that could result in mastery of the environment (Veeriah et al., 2018). The resulting behaviours are often more structured and directed than undirected exploration strategies like ϵ -greedy or softmax sampling.

Transfer. Options discovered through intrinsically motivated factor-based methods are transferable because they target changes in individual state variables or abstract subspaces. For example, in HEXQ, an option that changes a frequently occurring variable—like an agent’s position—can be reused across many contexts where that variable matters, regardless of the values of other variables. Similarly, in methods based on causal abstraction (Jonsson and Barto, 2006; Vigorito and Barto, 2008), options are constructed to affect only a specific part of the environment while assuming other parts are stable or independently controllable. This modularity reduces interference between options because each option specializes in a different part of the environment, which additionally encourages compositionality in behaviour space. As a result, once such an option is learned, it can be reused across multiple tasks or contexts without retraining, greatly improving sample efficiency.

Opportunities for Research.

- *Generalize factored approaches to large observation spaces.* Factored approaches like HEXQ assume access to explicit state variables, limiting their applicability to high-dimensional observation spaces such as images or sensor data. Recent works (Bagaria et al., 2026; Rodriguez-Sanchez et al., 2025; Kumar et al., 2025a; Higgins et al., 2016; Kim and Mnih, 2018) demonstrate some promising initial results, but significant challenges remain in automatically discovering meaningful factors without domain knowledge. Future research could focus on developing methods that identify relevant features and their causal dependencies in continuous, high-dimensional spaces while maintaining the transferability and composability benefits of factored approaches.
- *Improved estimates of novelty.* While count-based novelty measures work well in discrete spaces, they struggle in continuous environments where exact state revisitation is unlikely. Recent advances (Bellemare et al., 2016b; Burda et al., 2019; Machado et al., 2020; Lobel et al., 2023; Guo et al., 2022) provide neural network-based alternatives, but fundamental challenges persist in distinguishing meaningful novelty from environmental

stochasticity (sometimes colloquially called the “noisy TV” problem). Future work should develop more robust novelty estimation methods that can maintain exploration incentives across different timescales, handle function approximation errors gracefully, and integrate structural priors about environment dynamics to focus on causally relevant state changes.

- *Connections to other discovery techniques.* Intrinsic motivation approaches have developed largely independently from other option discovery methods like graph clustering, empowerment, and spectral decomposition techniques. However, recent theoretical work suggests that some of these approaches may be unified under information-theoretic frameworks (Achiam et al., 2018). Establishing formal connections between intrinsic motivation and other discovery paradigms (for example, the ROD cycle; Machado et al., 2023) could enable hybrid approaches that leverage the complementary strengths of different methods.

4.9 Directly Optimizing for the Benefits of Hierarchical Reinforcement Learning

Many of the option discovery methods that we have discussed so far rely on *proxy* objectives; these objectives include finding bottleneck states, empowerment maximization, more reliable composability, and so on. The intuition is that if the agent had options that maximized these proxy objectives, it would unlock agent-level capabilities such as effective exploration, credit assignment, or transfer. Indeed, these methods often show empirical success in some scenarios, but the formal connection between these proxy objectives and the overall objectives of the agent is unclear (Solway et al., 2014). For example, options that target bottleneck states are empirically useful in some tasks, but what kind of performance can we expect from the same technique in an entirely different problem? In fact, several papers have shown that not all skills are created equal—that is, options that are perfectly suited for a particular task might severely hurt agent-level objectives in other tasks (Jong et al., 2008; Solway et al., 2014). To address this gap, a class of methods—initiated by Solway et al. (2014)—has sought to discover options with precise guarantees on agent-level objectives. These methods explicitly state the performance criterion of the agent and then derive an algorithm that discovers options with bounded loss on that criterion.

In the language of Algorithm 1, these methods share a common structure: `OPTIONSPECIFICATION` is formulated as an explicit optimization problem over a precisely defined agent-level objective—such as planning time, cover time, or sample complexity during transfer. The option objectives Ω are the solution to this optimization, and `OPTIONLEARNING` then trains option policies to achieve them, typically using standard methods. What distinguishes this family is that the specification objective is not a proxy: it is a formally defined quantity that is either the agent-level objective itself or provably related to it. Most methods in this section solve `OPTIONSPECIFICATION` as a one-time combinatorial optimization before any learning or deployment, similar in phasing to some of the bottleneck methods of Section 4.1. However, whereas bottleneck methods use heuristic criteria to identify subgoals, these methods derive their subgoals from a well-defined performance criterion—often proving NP-hardness of the resulting optimization and providing approximation algorithms with bounded suboptimality. We organize the discussion by the agent-level objective being optimized: planning efficiency, exploration, credit assignment, and transfer.

4.9.1 BENEFITS AND OPPORTUNITIES

Planning. In the planning context, option discovery can be framed as the search for a set of options that minimizes the *planning time*—defined as the number of iterations a planning algorithm (e.g., value iteration) takes to approximate the optimal value function v^* within some accuracy ϵ (Silver and Ciosek, 2012; Jinnai et al., 2019a). Here, OPTIONSPECIFICATION amounts to solving a combinatorial optimization: find at most k options that minimize the number of value iteration steps needed to reach ϵ -accuracy. Formally, given a maximum allowable value error $\max_{s \in \mathcal{S}} |v^*(s) - \hat{v}(s)| \leq \epsilon$, the goal is to find a set of at most k options $\mathcal{O}$ that minimizes L_ϵ , the number of iterations needed to reach this accuracy:

$$\min_{\mathcal{O}} L_\epsilon \quad \text{s.t. } |\mathcal{O}| \leq k. \quad (73)$$

Jinnai et al. (2019a) prove that this problem is NP-hard, even in deterministic tabular MDPs. They introduce approximation algorithms with provable guarantees, but their results are limited to *point options*—options that initiate and terminate in a single state (Jinnai et al., 2019a).

While the above method minimizes worst-case planning time, Average Options (Ivanov et al., 2025) focuses instead on minimizing the expected planning time across a distribution of tasks. These tasks share the same transition dynamics but differ in their start and goal states. The idea is to discover options that reduce the expected cost of reaching any state from any other:

$$\arg \min_{\mathcal{O}} d_{\mathcal{O}}(G) = \arg \min_{\mathcal{O}} \sum_{s \in \mathcal{S}} \sum_{s' \in \mathcal{S}} d_{\mathcal{O}}(s, s'), \quad (74)$$

where $d_{\mathcal{O}}(s, s')$ is a non-symmetric distance metric (e.g., shortest path length) in the MDP graph augmented with options $\mathcal{O}$; such an augmentation adds edges to the graph while leaving nodes unchanged. Like the worst-case version, this problem is also NP-hard. However, by reducing it to the well-studied k -medians with penalties problem in graph theory (Meyerson and Tagiku, 2009), Ivanov et al. (2025) derive efficient approximation algorithms with bounded suboptimality. Planning can also be sped up using options in the single-task setting: Wan and Sutton (2022) present an option discovery algorithm that seeks options that maximize reward—similar to option-critic (Harb et al., 2018)—but reduces the number of options available at different states to reduce planning time.

Exploration. In the context of exploration, Jinnai et al. (2019c) formalize the performance criterion of the agent as minimizing the number of steps needed for a policy to visit every state (as a proxy for discovering some unknown reward). Here, OPTIONSPECIFICATION seeks options that minimize the expected cover time of a random walk on the augmented MDP graph. They show that this performance criterion is related to the graph-theoretic property of *cover time*, which measures how many steps a random walk needs to visit every state in a graph. To define cover time, we first need the *hitting time*. Given a Markov chain $\{X_t\}$ over states V , the hitting time from state i to state j is the first time the chain visits j when started from i :

$$H_{ij} = \inf\{t \geq 1 : X_t = j \mid X_0 = i\}. \quad (75)$$

The cover time from state i is the time needed to visit every state at least once:

$$C_i = \max_{j \in V} H_{ij}. \quad (76)$$

Both H_{ij} and C_i are random variables; the quantity of interest is the expected cover time $\mathbb{E}[C(G)]$, which bounds how long a random walk takes, in the worst case over starting states, to visit every state in the graph and hence to discover an unknown rewarding state (Jinnai et al., 2019c). Jinnai et al. (2019c) show that the expected cover time $\mathbb{E}[C_i]$ can be most effectively reduced by creating an option that connects the two states that are furthest apart according to the second eigenvector of the graph Laplacian (see Equation 22); this result provides a formal justification for the eigenoption heuristic discussed in Section 4.2. Jinnai et al. (2019c) also show that finding options that minimize cover time in a graph is NP-hard; but, they provide an approximation algorithm that minimizes an upper bound on the expected cover time. This method was later extended to continuous environments using deep learning-based approximations of the graph Laplacian (Jinnai et al., 2020; Wu et al., 2019), further strengthening the connections to the eigenoptions literature (Machado et al., 2017, 2023; Klissarov and Machado, 2023) (c.f. Section 4.2).

Credit Assignment. As discussed earlier, options can accelerate policy evaluation by enabling value updates that span multiple steps, rather than progressing one step at a time. Bacon and Precup (2016) formalize this intuition using the lens of *matrix splitting*, a technique from numerical linear algebra that speeds up the solution of linear systems. In this case, OPTIONSPECIFICATION seeks a set of options whose induced dynamics minimize the spectral radius of an iteration matrix, thereby maximizing the rate at which iterative policy evaluation converges. In their view, each set of options defines a modified Bellman operator that can be interpreted as a *preconditioned* version of the original policy evaluation problem. Recall that the Bellman expectation equation for a fixed policy π is:

$$\mathbf{v} = \mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v}, \quad (77)$$

where $\mathbf{v} \in \mathbb{R}^{|S|}$ is the value function, $\mathbf{r}_\pi$ is the expected reward vector, and $\mathbf{P}_\pi$ is the transition matrix under policy π . This is a linear system of the form $\mathbf{A}\mathbf{v} = \mathbf{b}$, with $\mathbf{A} = \mathbf{I} - \gamma \mathbf{P}_\pi$ and $\mathbf{b} = \mathbf{r}_\pi$. Planning with options induces a *matrix splitting* $\mathbf{A} = \mathbf{M} - \mathbf{N}$ (Varga, 2000), leading to an iterative update of the form:

$$\mathbf{v}_{k+1} = \mathbf{M}^{-1}\mathbf{N}\mathbf{v}_k + \mathbf{M}^{-1}\mathbf{b}. \quad (78)$$

In this formulation, the matrix $\mathbf{M}$ reflects the dynamics induced by the options and is chosen to be easy to apply and invert; the remaining part $\mathbf{N}$ captures what is not directly handled by the options. The matrix $\mathbf{M}^{-1}\mathbf{N}$ is the *iteration matrix*, as it governs how the current value estimate $\mathbf{v}_k$ influences the next one $\mathbf{v}_{k+1}$. This kind of transformation is known as *preconditioning*: a way of rewriting the problem so that the resulting iterative updates converge more quickly. The speed of convergence is governed by the *spectral radius* of the iteration matrix $\mathbf{M}^{-1}\mathbf{N}$ —its largest eigenvalue in absolute value. A smaller spectral radius means that errors shrink faster with each iteration. From this perspective, a good set of options is one whose induced splitting minimizes the spectral radius of $\mathbf{M}^{-1}\mathbf{N}$, enabling value information to propagate more efficiently. While Bacon and Precup (2016) do not

introduce a concrete option discovery algorithm, they offer a powerful design principle: discover options that act as preconditioners for value propagation. This opens the door to leveraging ideas from numerical linear algebra in option discovery.

Transfer. In the context of transfer, Solway et al. (2014) define the optimal set of options as those that maximize the efficiency with which an agent can learn the optimal policy for other, possibly unseen, sets of tasks. Here, `OPTIONSPECIFICATION` is an optimization over option sets that maximizes Bayesian model evidence under the distribution of tasks the agent expects to encounter. They show that a hierarchy maximizing model evidence also provably minimizes the geometric mean of the number of samples needed to find the optimal policy for any task in the given task distribution. Brunskill and Li (2014) consider a similar formulation of the option transfer problem: given interaction data from a set of tasks, how can an agent learn options that minimize the sample complexity of learning in a future stream of tasks? They find that this problem is at least as hard as the set cover problem in operations research, and is thus also NP-hard. They use a greedy approximation algorithm for option discovery and evaluate it empirically in a tabular MDP.

Opportunities for Research.

- *Guarantees in more general settings.* The papers discussed in this section emphasize the importance of formally stating the objective of option discovery and relating that to the overall objectives of the agent. However, this research is still nascent, and more work exploring this subject is needed. For instance, can we develop formal algorithms that bound planning time without needing the assumption of point options (Jinnai et al., 2019a)? Can we bound planning time or cover time when using function approximation? Although Brunskill and Li (2014) derive an algorithm to minimize sample complexity during transfer, the greedy approximation algorithm they present does not bound sample complexity; future work could extend their theoretical results to bound the performance of the greedy approximation algorithm.
- *Online and incremental option discovery with guarantees.* Most algorithms discussed in this section solve a combinatorial optimization offline, assuming access to a known MDP graph or to enough interaction data to construct one (Jinnai et al., 2019a; Ivanov et al., 2025). A few methods do construct options incrementally—for instance, Jinnai et al. (2019c) add options one at a time, each chosen to further reduce an upper bound on the expected cover time—but this incremental construction still operates on a fully known state-space graph. Extensions to larger or continuous state spaces do exist: Jinnai et al. (2020) scale covering options using deep learning-based approximations of the graph Laplacian, and Ivanov et al. (2025) adapt their planning-time objective to continuous domains by first discretizing the state space offline. In both cases, however, the formal guarantees of the tabular setting either do not carry over or apply only to the discretized proxy of the underlying problem rather than to the true MDP. An important open question is therefore whether any of the objectives considered in this section—planning time, cover time, or sample complexity during transfer—admit algorithms that discover and refine options as the agent collects data *while preserving* formal guarantees on the corresponding agent-level objective.

- *Stronger and more practically relevant guarantees.* The guarantees currently available in this literature are typically worst-case over all MDPs of a given size, and the algorithms producing them usually scale with the size of the state space (Ivanov et al., 2025). Can we obtain *instance-dependent* approximation ratios that adapt to structural properties of a given MDP—such as its diameter, bottleneck structure, or spectral gap—rather than worst-case ratios over all MDPs? Brunskill and Li (2014) provide PAC-style sample-complexity bounds for option discovery in the transfer setting, but analogous guarantees are missing for exploration. Fruit and Lazaric (2017) derive regret bounds for exploring with a *fixed* option set; can analogous guarantees be obtained when the options themselves are being discovered online, as they are for flat exploration algorithms (Kearns and Singh, 2002; Amortila et al., 2024)? And on the scaling side, can we characterize how the quality of discovered options degrades with the expressivity of a function approximator, the size of the option set, or the number of tasks in a transfer distribution?

4.9.2 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

In the language of Algorithm 1, the methods in this section all formalize OPTIONSPECIFICATION as an explicit optimization problem over a well-defined agent-level objective, rather than relying on a proxy. The output Ω is a set of subgoal states and corresponding option reward functions, just as in bottleneck discovery (Algorithm 2), but the subgoals are selected by optimizing an agent-level criterion rather than a graph-theoretic heuristic. OPTIONLEARNING and POLICYIMPROVEMENT are typically standard and receive less attention in this literature.

Two structural features distinguish these methods from those discussed earlier. First, the optimization that determines Ω considers interactions between options: the value of including any single option objective in Ω depends on what other objectives are already present. In bottleneck methods (Section 4.1), each element of Ω is typically scored independently—for example, the betweenness score of a state does not depend on what other options exist. Here, by contrast, adding an edge to the graph that connects two distant states is valuable only if no existing option already provides a similar shortcut. This set-level dependence is precisely what makes these optimization problems NP-hard and necessitates approximation algorithms. Second, many approaches in this section adopt a separated pipeline: the optimization over Ω is solved once given a model or dataset, and the resulting subgoals are fixed before deployment. In such cases, OPTIONSPECIFICATION runs once, which may limit adaptability when new regions of the state space are discovered. However, the cover-time minimization algorithm of Jinnai et al. (2019c) has also been applied in an online setting, where OPTIONSPECIFICATION is re-invoked periodically as the agent gathers more experience and refines its estimate of the transition graph, interleaving OPTIONSPECIFICATION with OPTIONLEARNING and INTERACT.

5. Discovery through Offline Datasets

Offline RL (or batch RL) aims to learn policies from pre-collected datasets, avoiding active data collection. This enables scalable deployment in domains such as robotics, autonomous driving, education, and healthcare (Levine et al., 2020), where interaction data can be

difficult to obtain. Similarly, *offline skill discovery* leverages these datasets to extract temporally abstract behaviours that can later serve as high-level primitives (in either offline or online RL) to accelerate learning.

In this section, to facilitate the discovery of skills, we assume access to an offline dataset $\mathcal{D} = \{\tau_i\}_{i=1}^N$, where τ_i is a trajectory generated through interaction with the environment. When all components are observed, a trajectory can be written as $\tau_i = (s_t^i, a_t^i, r_t^i)_{t=1}^T$. Depending on the method and dataset, however, a trajectory may contain only a subset of these components. The trajectories may be expert demonstrations or data collected by one or more behaviour policies. For example, many methods in Section 5.1 do not require rewards, and some do not require actions (Kim et al., 2019).

A related line of work learns reusable representations of environment dynamics, often from offline data, without explicitly defining temporally extended skills. One important family is based on successor representations and successor features. These representations separate expected future state occupancies or feature accumulations from the task-specific reward, allowing previously learned knowledge to be reused when the reward changes (Dayan, 1993; Barreto et al., 2017, 2020). Universal successor features (Ma et al., 2020), forward-backward representations (Touati et al., 2023; Tirinzoni et al., 2025), and successor-measure-based auxiliary tasks (Farebrother et al., 2023) develop this principle in different ways. Other approaches capture reusable structure using random features (Chen et al., 2023), Hilbert-space representations (Park et al., 2024b), or Laplacian representations (Shehmar et al., 2026; Chandrasekar and Machado, 2026). Because these methods do not necessarily define temporally extended skills, we do not discuss them in detail.

5.1 Variational Inference of Skill Latents

Offline datasets often contain recurring, temporally extended behaviours without labels for either the behaviours or their boundaries. For example, a trajectory from a robotic manipulation task may include reaching for an object, grasping it, and transporting it to a target location without indicating where one behaviour ends and another begins. To uncover this structure, variational-inference methods use an encoder to map trajectory segments to latent codes and a decoder to reconstruct the observed trajectories. Following common usage in the literature, we refer to each latent code as a *skill* throughout this section.¹³

These methods usually rely on *unlabeled experiences*, $\tau = (s_t, a_t)_{t=1}^T$ —that is, data collected without explicit reward feedback—and in some cases, even excludes actions (Kim et al., 2019). This is often referred to as *unsupervised skill discovery* (Eysenbach et al., 2019). To accommodate different formulations in the literature, we associate each trajectory τ with a latent skill sequence ζ :

$$\zeta = (z_t, b_t)_{t=1}^T, \quad z_t \in \mathbb{R}^d, \quad b_t \in \{0, 1\}, \quad (79)$$

where z_t represents the skill active at time t and b_t is a boundary signal that indicates when a skill starts or ends, i.e. the analogue of an option-termination signal.¹⁴

13. Under the formalism of Definition 4 and Section 3.2.3, the latent code is an option identifier: it indexes the corresponding option rather than denoting the complete option itself.

14. Both z and b can be represented in different forms across methods. For example, Kipf et al. (2019) uses $b_m \in \mathbb{Z}$ to denote the position of the m -th boundary, while some methods omit explicit boundary variables (Ajay et al., 2021).

Let $p_\phi(\tau, \zeta)$ denote a generative model over a trajectory τ and its latent skill sequence ζ , parameterized by ϕ . The model is learned by maximizing the marginal log-likelihood of the trajectories in the offline dataset $\mathcal{D}$:

$$J(\phi) = \mathbb{E}_{\tau \sim \mathcal{D}} [\log p_\phi(\tau)]. \quad (80)$$

Here the term $p_\phi(\tau) = \int p_\phi(\tau, \zeta) d\zeta$ has already *marginalized* the latent skill sequence ζ , so maximizing $J(\phi)$ encourages the model to explain the observed trajectories without fixing any particular skills in advance. Because marginalizing over all possible ζ is usually intractable, variational methods introduce an approximate posterior $q_\phi(\zeta | \tau)$ and apply Jensen’s inequality to derive the *evidence lower bound* (ELBO), a tractable lower bound on $\log p_\theta(\tau)$:

$$\begin{aligned} \log p_\phi(\tau) &= \log \int q_\phi(\zeta | \tau) \frac{p_\phi(\tau, \zeta)}{q_\phi(\zeta | \tau)} d\zeta \\ &\geq \mathbb{E}_{q_\phi(\zeta | \tau)} [\log p_\phi(\tau, \zeta) - \log q_\phi(\zeta | \tau)] \\ &= \mathbb{E}_{q_\phi} [\log p_\phi(\tau | \zeta)] - D_{\text{KL}}(q_\phi(\zeta | \tau) \| p_\phi(\zeta)). \end{aligned} \quad (81)$$

Averaging over $\tau \sim \mathcal{D}$, and introducing the β -weight¹⁵ (Higgins et al., 2016) which balances reconstruction and regularization terms yields the training objective:

$$J_{\text{ELBO}}(\phi) = \underbrace{\mathbb{E}_{\tau \sim \mathcal{D}, \zeta \sim q_\phi(\zeta | \tau)} [\log p_\phi(\tau | \zeta)]}_{\text{reconstruction}} - \beta \underbrace{\mathbb{E}_{\tau \sim \mathcal{D}} D_{\text{KL}}(q_\phi(\zeta | \tau) \| p_\phi(\zeta))}_{\text{regularization}}, \quad (82)$$

which is *maximized* with respect to ϕ . The first term encourages ζ to capture the latent skill structure that makes τ likely under $p_\phi(\tau | \zeta)$. The second term *regularizes* those encodings toward the prior, encouraging the emergence of a compact skill space. In practice, three distinct models are employed whose parameters are jointly denoted by ϕ :

- **Prior** $p_\phi(\zeta)$: defines a prior distribution over latent skill sequences. A common choice is a fixed, factorized prior (e.g., unit Gaussian for z_t and Bernoulli for b_t), but the prior can instead be endowed with learnable parameters and conditioned on the current state, yielding $p_\phi(\zeta | s)$ (Ajay et al., 2021; Nam et al., 2022). This can also serve as a prior on the policy over skills (i.e., a high-level policy), $\mu_\kappa(z | s)$.
- **Decoder** $p_\phi(\tau | \zeta)$: models the probability of a trajectory conditioned on a given skill sequence. In many works, this likelihood is factorized over timesteps and parameterized as a skill-conditioned policy,

$$p(\tau | \zeta) = p(s_1) \prod_{t=1}^T p(s_{t+1} | s_t, a_t) \pi_\theta(a_t | s_t, \zeta) \propto \prod_{t=1}^T \pi_\theta(a_t | s_t, \zeta), \quad (83)$$

so that the decoder can reconstruct only the action sequence conditioned on states and skills (Kipf et al., 2019; Ajay et al., 2021; Pertsch et al., 2021; Nam et al., 2022).

15. We slightly abuse the notation here, where usually in this paper β refers to the termination function.

Under this parameterization, Equation 82 becomes:

$$J_{\text{ELBO}}(\phi, \theta) = \underbrace{\mathbb{E}_{\tau \sim \mathcal{D}, \zeta \sim q_\phi(\zeta | \tau)} \left[\sum_{t=1}^T \log \pi_\theta(a_t | s_t, \zeta) \right]}_{\text{action-reconstruction}} - \beta \underbrace{\mathbb{E}_{\tau \sim \mathcal{D}} D_{\text{KL}}(q_\phi(\zeta | \tau) \| p_\phi(\zeta))}_{\text{regularization}}. \quad (84)$$

- **Encoder** $q_\phi(\zeta | \tau)$: given an observed trajectory, returns a distribution over the skill sequence that likely produced it.

Variational-autoencoder (VAE)-based methods (Kingma and Welling, 2014) are the main focus of this subsection: they train an encoder $q_\phi(\zeta | \tau)$ that predicts a distribution over ζ from each trajectory and optimize the ELBO in Equation 82. Related latent-variable methods share the same high-level goal of inferring a latent sequence ζ that explains the observed trajectories, but differ in how ζ is inferred and which training objective is used. Expectation-Gradient methods (Fox et al., 2017; Krishnan et al., 2017) instead begin with unsegmented trajectories in which the active option and the option boundaries are unknown. At each timestep, they estimate a probability distribution over the active option, together with the probability that the previous option terminated and a new option was selected. They then use these probabilities to update the option policies, termination functions, and high-level policy so that the model assigns higher likelihood to the observed trajectories. Adversarial approaches such as Directed-Info GAIL (Sharma et al., 2018) train a discriminator to distinguish policy-generated state-action pairs from demonstration, while using an auxiliary objective to encourage different skills to induce distinguishable behaviours.

The learned skills naturally support a hierarchy. In such works, a low-level controller, $\pi_\theta(a_t | s_t, z_t)$, is typically trained offline via behavioural cloning to execute any given skill, while a high-level policy, $\mu_\kappa(z_t | s_t)$, is subsequently optimized, either online (Pertsch et al., 2021; Nam et al., 2022; Salter et al., 2022), or with offline RL (Ajay et al., 2021). Learned skills can also augment the primitive action space, expanding the agent’s control repertoire (Fox et al., 2017; Kipf et al., 2019; Jiang et al., 2022), or be used to construct pseudo-reward signals by measuring the distance between each transition’s latent embedding and calibrated expert behaviour (Liu et al., 2023b).

Beyond pure likelihood maximization, it is also common to add a **compression** regularizer grounded in the *minimum description length* (MDL) principle (Rissanen, 1978; Thrun and Schwartz, 1994). MDL prefers the model that can be transmitted with the fewest total bits of (i) the model parameters and (ii) the data encoded through that model. Viewing the latent codes as skill identifiers and boundaries (Equation 79), the model parameters are the decoder (and prior if parameterized), and the data are the offline trajectories; hence, a concise skill set shortens the overall description length.

The *bits-back* coding argument (Hinton and Zemel, 1993; Honkela and Valpola, 2004; Zhang et al., 2021b) shows that maximizing the ELBO (Equation 82) approximately minimizes the description length, but with an ill-chosen prior $p(\zeta)$, the optimum can collapse to a degenerate representation (e.g., a single skill encoding that simply mirrors the observations). To avoid this, LOVE (Jiang et al., 2022) augments the ELBO with an MDL-inspired information-cost term that explicitly penalizes skills increasing the expected code length of

transmitting trajectories, yielding a representation that is both *informative* (high ELBO) and *economical* (low description length):

$$\min_{\phi} L_{\text{DL}}(\phi) \quad \text{s.t.} \quad J_{\text{ELBO}}(\phi) \geq C, \quad (85)$$

$$L_{\text{DL}}(\phi) = \mathbb{E}_{\tau \sim \mathcal{D}, \{b_t, z_t\} \sim q_{\phi}(\cdot | \tau)} \left[- \sum_{t=1}^T b_t \log p_z^*(z_t; \phi) \right], \quad (86)$$

where $b_t = 1$ indicates the start of a new skill at time t , and C is a constant. The optimal prior on z , p_z^* , that minimizes the expected description length, is defined by:

$$p_z^*(z; \phi) = \frac{\mathbb{E}_{\tau \sim \mathcal{D}, \{b_t, z_t\} \sim q_{\phi}(\cdot | \tau)} \left[\sum_{t=1}^T b_t \mathbb{1}_{\{z_t=z\}} \right]}{\mathbb{E}_{\tau \sim \mathcal{D}, \{b_t, z_t\} \sim q_{\phi}(\cdot | \tau)} \left[\sum_{t=1}^T b_t \right]}, \quad (87)$$

with $\mathbb{1}$ denoting the indicator function. Intuitively, $\mathcal{L}_{\text{DL}}$ penalizes having too many skill boundaries and distinct skill choices, driving the method toward a concise skill decomposition, and longer skills encompassing common structures are generally favored to avoid the degenerate solution where each skill represents a single action. Salter et al. (2022) instead leverages the concept of *bottleneck* option by introducing a *predictability* objective, encouraging option-level transitions to be predictable. The authors show that maximizing this predictability reduces the conditional entropy and thus the optimal code length, and this objective is equivalent to applying the MDL principle.

Vlastelica et al. (2023) cast offline skill discovery as **empowerment maximization**, presented in Section 4.4, under an imitation constraint. To ensure that each skill remains faithful to the offline demonstrations, they constrain the divergence between the induced state occupancy and the state occupancy d_E from a skill-independent expert dataset, resulting in a constrained optimization problem:

$$\max_{\{\pi_z\}, q_{\phi}} \mathbb{E}_{z \sim p(z), s \sim d_{\pi}} [\log q_{\phi}(z | s)] \quad \text{s.t.} \quad D_{\text{KL}}(d_{\pi} \| d_E) \leq \varepsilon, \quad \forall z. \quad (88)$$

Here, d_{π} denotes the state-occupancy measure of the skill-conditioned policy, estimated in the offline setting via density-based learning. The term q_{ϕ} represents a skill discriminator that tightens a variational lower bound on the mutual information between skills and states. It simultaneously *diversifies* behaviours by maximizing the lower bound on the mutual information between states and skills, and *regularizes* them by penalizing departures from the expert state distribution d_E .

5.1.1 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

When instantiating Algorithm 11, `OPTIONSPECIFICATION` typically runs only once and Ω is fixed thereafter. Design Principle 1 in Section 3.2.2 does not typically hold here; instead, a single shared objective is used, and the method performs single-level optimization, simultaneously inferring z and b and learning the skill-conditioned policy within `OPTIONLEARNING`.

Algorithm 11 Variational Inference of Skill Latents: Instantiation of Algorithm 1.

Require: Offline dataset $\mathcal{D}$, prior $p_\phi(\zeta)$.

- 1: Initialize model parameters ϕ, θ .
 - 2: **procedure** OPTIONSPECIFICATION($\mathcal{O}, \mathcal{D}$)
 - 3: Define (the structure of) the latent skill variables $\zeta = (z_t, b_t)_{t=1}^T$.
 - 4: Define the objective $\Omega = J_{\text{ELBO}}(\phi, \theta)$.
 $\triangleright$ Eq. 82, can also include MDL or other objectives, from Eq. 85–Eq. 88.
 - 5: **return** Ω $\triangleright$ Defines what latent skills should explain
 - 6: **end procedure**
 - 7: **procedure** OPTIONLEARNING($\mathcal{O}, \mathcal{D}, \Omega$)
 - 8: Sample $\tau \sim \mathcal{D}$ and infer $\zeta \sim q_\phi(\zeta \mid \tau)$.
 - 9: Update ϕ, θ to maximize Ω .
 - 10: Construct $\mathcal{O}$ implicitly from learned skills z and boundary signals b .
 $\triangleright$ Each z induces an option via $\pi_\theta(a \mid s, z)$ with termination $\beta(s, z) = \mathbb{1}(b = 1)$
 - 11: **return** $\mathcal{O}$
 - 12: **end procedure**
-

5.1.2 BENEFITS AND OPPORTUNITIES

Credit Assignment. By extracting hierarchical structure from offline datasets, agents can break complex trajectories into more manageable subgoals. This segmentation makes it easier to understand why certain results occur, as it allows each outcome—such as achieving a subgoal—to be more directly linked to the specific actions and conditions that produced it. In other words, by focusing on a sequence of subgoals rather than the full sequence of primitive actions, the learning algorithm can more easily attribute success or failure to specific decisions or events. For example, Kipf et al. (2019) tackle credit assignment in maze-like environments with delayed feedback. Their method infers segment boundaries $q(b_i \mid s, a)$, with $b_i \in [1, T + 1]$ functioning similarly to option termination functions, and encodes each segment using $q(z_i \mid s, a)$ as latent skill (subgoal) descriptors. This segmentation captures subgoal structure, facilitating effective credit assignment across subgoals when applying the skills in sparse-reward settings. Similarly, Kim et al. (2019) improve credit assignment in goal-oriented navigation tasks by decomposing action-free trajectories into subsequences by inferring skill descriptors z_t and binary termination signals $b_t \in \{0, 1\}$.

Transfer. By discovering skills from offline datasets, agents develop a foundational set of versatile competencies. These competencies can then be transferred to new tasks with minimal adjustment. Pertsch et al. (2021) show promising results on transferring skills obtained in the offline dataset to more complex simulated robotic tasks unseen in the dataset (e.g., maze navigation with larger maps in evaluation). Similarly, Jiang et al. (2022) and Salter et al. (2022) show that by optimizing a compression objective, in addition to the reconstruction one, the discovered skills help transfer across multiple tasks. Nam et al. (2022) demonstrate that by meta-training a high-level policy, $\pi_\theta(z_t \mid s_t, e)$, where e is a task

encoding, and executing a low-level policy, $\pi_\theta(a_t | s_t, z_t)$, which is learned via behavioural cloning, the agent can solve a wide range of new tasks in a meta-RL setting.

Exploration. When reusing the offline discovered skills for online interaction, this can reduce the difficulty of exploration since the agent can quickly apply well-tested, pre-learned behaviours rather than learning them through trial-and-error in real-time. Fox et al. (2017) show promising exploration results in a simple four-room domain by augmenting the action space with discovered parameterized options. Salter et al. (2022) show that the learned temporally compressed bottleneck options are beneficial for exploration in maze-like environments with delayed rewards.

Avoiding Distributional Shift. In offline RL, a learned policy may select actions with little or no support in the fixed dataset, for which value estimates can be unreliable. Ajay et al. (2021) leverage offline skill discovery to mitigate this issue. Their approach encodes short trajectories (e.g., every K steps) from the dataset into a skill code z . By maximizing the log-likelihood of actions in trajectories τ , conditioned on states s_t and skill descriptors z , the method captures recurring behaviours present in the data. The offline dataset can then be enhanced using z , and a high-level policy, $\pi_\theta(z | s)$, can be derived by off-the-shelf offline RL algorithms. The authors show that such a temporal structure reduces compounding errors for extrapolating out-of-distribution actions in offline RL.

Opportunities for Research.

- *Optimization challenges.* Evident in some studies (Jiang et al., 2022), optimization challenges can lead to degraded skill quality as poor local optima corresponding to degenerate skill decompositions can be empirically hard to escape. Additionally, the under-utilization of reward signals in existing datasets creates an opportunity to further refine learned skills, and incorporating offline RL methods—rather than relying solely on reconstruction-based approaches—into HRL may unlock greater performance gains, as Hu and Leung (2023) provide positive theoretical results on sample efficiency.
- *Broader scope of test environments.* Additionally, many methods in this field primarily validate their concepts on simulated robotic navigation tasks, which typically involve deterministic transitions and rewards (Gao et al., 2024), and often favor specific inductive biases. A natural extension would be scaling this paradigm to real-world, image-based tasks, or other practical applications with different properties.

5.2 Hindsight Subgoal Relabeling

In Section 5.1, we discussed the methods that infer the skill descriptors, usually characterized by latent codes or skill embeddings. In this section, we explore methods that identify and relabel subgoals within an offline dataset, effectively leveraging existing trajectories to learn how to achieve subgoals (Kaelbling, 1993). Offline experiences offer valuable insights into identifying subgoals, which can be viewed as milestones or waypoints for accomplishing a task (Gupta et al., 2019; Park et al., 2024a), or abstracting bottleneck states to make a good partition of the state space (Paul et al., 2019). This is conceptually similar to hindsight experience replay (Andrychowicz et al., 2017) we discussed in Section 4.6, which relabels the final or intermediate states reached in a trajectory as if they were the intended goals.

In an exemplar work by Paul et al. (2019), a reward-free trajectory dataset, $\mathcal{D} = \{(s_1^i, a_1^i, \dots, s_T^i)\}_{i=1}^N$, is segmented into an *ordered* list of n_g disjoint partitions, $G = \{1, \dots, n_g\}$, that serve as subgoals. The label $g_t^i = j$ indicates that state s_t^i is assigned to the j -th subgoal region.

Initial relabeling. Because the demonstrations do not provide these assignments, the method begins with a simple initial guess: each trajectory is divided into n_g consecutive portions of approximately equal duration, and the j -th portion is assigned label j :

$$g_t^i = j \quad \text{if and only if} \quad \left\lfloor \frac{(j-1)T}{n_g} \right\rfloor < t \leq \left\lfloor \frac{jT}{n_g} \right\rfloor, \quad j \in G. \quad (89)$$

Iterative refinement. Repeat the following two steps until the label change falls below a threshold. Alternating these steps until convergence yields a classifier μ_κ that both partitions the state space and respects the required ordering. Such a $\mu_\kappa(g \mid s)$ can provide information on whether a state is a bottleneck.

- *Learning step:* fit a classifier $\mu_\kappa(g \mid s)$ by cross-entropy on the current labels:

$$L(\kappa) = \mathbb{E}_{(s,g) \sim \mathcal{D}} [-\log \mu_\kappa(g \mid s)]. \quad (90)$$

- *Inference step:* enforce the trajectory order $1 \prec 2 \prec \dots \prec n_g$ with Dynamic Time Warping¹⁶ (Müller, 2007) over the sequence $(\mu_\kappa(g=1 \mid s_t^i), \dots, \mu_\kappa(g=n_g \mid s_t^i))$.

Potential function and intrinsic reward. The most probable class defines a potential $\Phi_\kappa(s) = \arg \max_g \mu_\kappa(g \mid s)$, representing the most likely subgoal index associated with state s . Based on this potential, an intrinsic reward, r' , that measures *progress* toward successive subgoals, is defined as:

$$r'(s, a, s') = \gamma \Phi_\kappa(s') - \Phi_\kappa(s), \quad (91)$$

where γ is the discount factor of the MDP.

Learning schedule. The policy π_θ is first initialized through behaviour cloning, after which online RL proceeds with the augmented reward $r + r'$. This phase exploits subgoal guidance to supply dense progress signals without additional expert interaction and still preserves the original optimum (Paul et al., 2019).

As another example, Relay Policy Learning (RPL) (Gupta et al., 2019) relabels all demonstration trajectories $\{\tau^i\}_{i=1}^N$, where $\tau^i = (s_1^i, a_1^i, \dots, s_T^i)$, to construct a low-level dataset $\mathcal{D}_\ell$ and a high-level dataset $\mathcal{D}_h$ (represented jointly as $\mathcal{D}$ in Algorithm 12):

$$\mathcal{D}_\ell = \{ (s_t, a_t, g_\ell) \mid g_\ell = s_{t+w}, 0 \leq t < T, 1 \leq w \leq W_\ell, t+w \leq T \}, \quad (92)$$

$$\mathcal{D}_h = \{ (s_t, g_\ell, g_h) \mid g_\ell = s_{t+\min(W_\ell, w)}, g_h = s_{t+w}, 0 \leq t < T, 1 \leq w \leq W_h, t+w \leq T \}. \quad (93)$$

Here, w is the number of timesteps between the current state and a selected future state, while W_ℓ and W_h are the maximum look-ahead windows used to construct low- and high-level

16. An algorithm that aligns two sequences by allowing non-uniform time shifts, so that similar patterns are matched even if they occur at different timesteps or speeds.

Algorithm 12 Hindsight Subgoal Relabeling: Instantiation of Algorithm 1.**Require:** Offline dataset $\mathcal{D}$, subgoal relabeling rule $\mathbb{S}$.

- 1: Initialize option set $\mathcal{O}$, low-level policy π_θ .
- 2: **procedure** INTERACT($E, \mathcal{O}, \mu, \mathcal{D}$)
- 3: Sample trajectory $\tau \sim \mathcal{D}$.
- 4: Relabel τ with subgoals using $\mathbb{S}$.
- 5: Construct relabeled dataset $\mathcal{D}'$.
- 6: **return** $\mathcal{D}'$ ▷ Offline setting: no new environment interaction
- 7: **end procedure**
- 8: **procedure** OPTIONSPECIFICATION($\mathcal{O}, \mathcal{D}$)
- 9: Define subgoals g (from relabeled states in $\mathcal{D}$ or from a learned classifier).
- 10: Define subgoal-conditioned objectives $\Omega = \{r^g\}$.
▷ e.g., goal-reaching reward or intrinsic reward from subgoal progress
- 11: **return** Ω
- 12: **end procedure**
- 13: **procedure** OPTIONLEARNING($\mathcal{O}, \mathcal{D}, \Omega$)
- 14: Update option policy $\pi_\theta(a \mid s, g)$ to maximize $r^g \in \Omega$.
- 15: Construct options $\mathcal{O}$ from learned goal-conditioned behaviours.
▷ $\forall s, \mathcal{I}(s) = 1; \beta(s) = \mathbb{1}\{s \text{ satisfies } g\}$
- 16: **return** $\mathcal{O}$
- 17: **end procedure**

training examples, respectively. RPL uses $W_\ell = 30$ and $W_h = 260$ in all experiments. Thus, $\mathcal{D}_\ell$ offers *short-horizon* examples (subgoal horizon $\leq W_\ell$) for training a low-level controller $\pi_\theta(a \mid s, g_\ell)$ to reach nearby states, whereas $\mathcal{D}_h$ pairs each *long-horizon* target g_h (up to W_h steps away) with a feasible intermediate subgoal g_ℓ , enabling hierarchical planning by a high-level goal-setter $\mu_\kappa(g_\ell \mid s, g_h)$. The imitation objective is:

$$\max_{\kappa, \theta} \mathbb{E}_{(s, a, g_\ell) \sim \mathcal{D}_\ell} [\log \pi_\theta(a \mid s, g_\ell)] + \mathbb{E}_{(s, g'_\ell, g_h) \sim \mathcal{D}_h} [\log \mu_\kappa(g'_\ell \mid s, g_h)].$$

During execution, every W_ℓ steps the high-level policy samples a new subgoal $g_\ell \sim \mu_\kappa(\cdot \mid s, g_h)$ and the low-level controller tracks it step-by-step until the next subgoal is issued.

5.2.1 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

Algorithm 12 has a similar algorithmic structure as Algorithm 9 for hindsight experience replay but differs in design details. Same as described in Section 4.6.1, we choose to put the relabeling process in INTERACT but conceptually it can be regarded as OPTIONSPECIFICATION. Note that the work by Paul et al. (2019) can be interpreted as an instance of option discovery where subgoal indices g , inferred via the classifier $\mu_\kappa(g \mid s)$, serve as option identifiers, inducing implicit options through the potential function $\Phi_\kappa(s)$ and the resulting intrinsic reward, rather than through explicit goal-conditioned policies as

in Gupta et al. (2019), which slightly differs from the description in Algorithm 12, and its OPTIONSPECIFICATION is similar to Algorithm 5.

5.2.2 BENEFITS AND OPPORTUNITIES

Credit Assignment. Subgoal relabeling decomposes complex tasks into manageable intermediate objectives by highlighting which actions contribute to reaching the final goal.

Paul et al. (2019) address credit assignment by constructing an intrinsic reward based on a subgoal policy $\mu_\kappa(g \mid s)$, which identifies a state’s progress toward a final goal. This classifier is trained via an EM-style procedure that enforces an ordering constraint over subgoal indices, ensuring that states later in a demonstration receive higher labels. The resulting potential function, $\Phi_\kappa(s) = \arg \max_g \mu_\kappa(g \mid s)$, induces a shaped reward (Equation 91) which provides dense feedback aligned with behavioural progress. This intrinsic signal facilitates temporal credit assignment by rewarding transitions that advance the agent through the learned subgoal structure, even when extrinsic rewards are sparse or delayed.

Alternatively, RPL (Gupta et al., 2019) addresses the credit assignment challenge from sparse, delayed rewards by relabeling demonstrations with overlapping sliding-window subgoals. It trains a goal-conditioned low-level policy to reach short-horizon targets, while a high-level policy selects subgoals. This hierarchical structure ensures that each low-level episode ends with an intrinsic success signal. As a result, external rewards propagate after at most one window. This transforms the long-horizon problem into a sequence of locally supervised updates, enabling faster and more stable credit assignment than flat or single-level baselines.

Similarly, Park et al. (2024a) relabel subgoals as the state s_{t+k} that lies exactly k steps ahead of the current state s_t . A high-level policy, $\mu_\kappa(s_{t+k} \mid s_t, g)$, proposes such waypoints conditioned on the ultimate goal g , while a low-level policy, $\pi_\theta(a_t \mid s_t, s_{t+k})$, outputs primitive actions that move the agent toward the subgoal. Both policies are optimized via a shared goal-conditioned value function V_ϕ : μ_κ maximizes $V_\phi(s_{t+k}, g)$, whereas π_θ maximizes $V_\phi(s_{t+1}, s_{t+k})$. Specifically, μ_κ is trained to choose subgoals s_{t+k} that maximize $V_\phi(s_{t+k}, g)$, while π_θ is trained to select actions that make the next state s_{t+1} have high value $V_\phi(s_{t+1}, s_{t+k})$ relative to the current subgoal. Because different subgoals induce much larger variations in V_ϕ than individual actions, the high-level receives a more reliable learning signal, and since π_θ queries V_ϕ only for nearby states where estimates are more accurate, the entire hierarchy is less susceptible to noise and approximation errors in the value function, resulting in a more robust policy and credit assignment.

Interpretability. Identifying subgoals within the offline dataset can provide insights into understanding the decision-making process. For example, Paul et al. (2019) present visualizations of the state space in robotic navigation tasks, such as AntMaze and AntTarget, demonstrating that the state space can be structurally partitioned using discovered subgoals. The structural decomposition is intuitively meaningful to humans, facilitating better understanding and verification.

Opportunities for Research.

- *Enhancing interpretability of decision making with interpretable subgoals.* Although Paul et al. (2019) show positive empirical results, current offline relabeling schemes select subgoals with limited transparency into why particular states are chosen or

how they steer the learned policy. Embedding interpretability or alignment objectives, such as attributing subgoal selection to human-understandable criteria, would not only clarify the decision rationale, but also foster trust and diagnosability.

- *Scaling to complex observations by identifying latent subgoals.* Researchers can extend offline subgoal relabeling to environments with high-dimensional inputs, such as images, language, or tactile data, with the work by Park et al. (2024a) as an example. To do so, methods could identify subgoal representations in some latent space that pinpoint meaningful milestones within these spaces. By focusing on compact embeddings, relabeling can remain effective even when raw observations are noisy, partial, or multimodal.

6. Discovery with Foundation Models

Agents that learn skills from scratch through environment interactions are directly exposed to the inherent complexities of the domains in which they operate. Such agents must learn from their stream of experience how to organize the collected data into meaningful chunks in order to derive a useful set of skills. To mitigate these challenges, we can instead build on prior knowledge contained in large pretrained models to guide the discovery of useful skills in complex environments. The starting assumption for such methods is that pretrained models contain knowledge about the environment of interest. Although this assumption may not always hold, it is likely applicable to many domains of interest and will grow as the training paradigm of LLMs expands in scope.

Simultaneously, an interesting feature of LLM-based methods is that, as these large models are based on human priors and are instantiated through natural language, the set of behaviours will generally be more interpretable. In fact, leveraging language, without using LLMs, has produced a prolific line of work (Shu et al., 2018; Bahdanau et al., 2018; Fu et al., 2019; Jiang et al., 2019; Colas et al., 2020a,b; Akakzia et al., 2021).¹⁷ These works underscore, amongst other features, the compositional nature of language. This quality makes it a particularly useful space to represent a variety of goals.

It is therefore natural to consider LLMs for HRL as they provide both useful inductive biases from pre-training on human data and a meaningful abstraction space through natural language. This connection is reinforced by the fact that **LLMs, by their very nature, can represent goal-conditioned policies, where goals are specified linguistically.**

As such, many recent works leverage LLMs to decompose tasks into subtasks (Pignatelli et al., 2024; Yang et al., 2025; Wang et al., 2024c), an operation done according to their pre-existing understanding of the task’s underlying structure. Another perspective is to use LLMs as a measure of interestingness to propose a curriculum of goals and tasks in open-ended domains (Colas et al., 2023; Zhang et al., 2024; Faldor et al., 2025; Wang et al., 2024b; Zala et al., 2024). The key to converting an LLM’s latent knowledge into a functional agent lies in efficiently learning the options required to execute the decomposed goals.

In this section, we investigate four families of methods that propose solutions to this problem. The first family consists of methods using embeddings from large pretrained models as representations from which option rewards are defined. Next, we present methods that use large pretrained models to provide feedback, in the form of rewards or preferences,

17. For an in-depth review of the use of language in RL, please refer to the work by Luketina et al. (2019).

for learning different skills. Building on the code generation capabilities of large models, we present two families of methods that write code to either craft reward functions to learn specific behaviours. Finally, as LLMs can be seen as goal-conditioned policies, we cover methods that use them directly to specify subgoals and achieve them.

Not all the presented methods in this section are hierarchical by nature, where hierarchical is defined as containing a high level policy that sequentially selects goals. In some cases, papers focus on defining reward for learning task-conditioned or skill-conditioned policies, without including the high level policy. Given our focus on methods for discovering temporal structure, we believe that learning from these methods is very valuable.

6.1 Embedding Similarity

As foundation models are trained on Internet-scale datasets, their embeddings contain useful structure for a variety of tasks. Such embeddings can be the result of contrastive pretraining on image and text pairs, for instance, the Contrastive Language-Image Pretraining (CLIP) encoder (Radford et al., 2021). Methods in this section leverage these embedding in pixel-based tasks to learn different behaviours specified through language. Let $\mathbf{w}_i \in \mathbb{R}^d$ represent the normalized feature vector (embedding) generated by the image encoder for the i -th image, I_i . Similarly, let $\mathbf{u}_j \in \mathbb{R}^d$ be the normalized feature vector generated by the text encoder for the j -th text, T_j . The similarity between image i and text j is computed using the cosine similarity (which simplifies to the dot product for normalized vectors):

$$C_{ij} = \mathbf{w}_i^\top \mathbf{u}_j. \quad (94)$$

These embeddings can then be used to represent image or language goals and define reward functions by taking the cosine similarity between the embeddings of the goal and the observation in which the agent is currently situated,

$$r^g(s) = \mathbf{w}(s)^\top \mathbf{u}(g). \quad (95)$$

This reward function is then maximized by a goal-conditioned policy interacting with an environment to learn behaviours that achieve the specified goals.

To obtain these vectors, the objective is formulated as minimizing a cross-entropy loss, applied symmetrically for both image-to-text and text-to-image prediction tasks. The loss for predicting the correct text caption for a given image i (considering all N text captions in the batch) is defined as:

$$L_{\text{image}_i} = -\log \frac{\exp(c_{ii}/e)}{\sum_{j=1}^N \exp(c_{ij}/e)}, \quad (96)$$

where e is the temperature hyperparameter. The loss for predicting the correct image for a given text caption i (considering all N images in the batch) is:

$$L_{\text{text}_i} = -\log \frac{\exp(c_{ii}/e)}{\sum_{j=1}^N \exp(c_{ji}/e)}. \quad (97)$$

Algorithm 13 MineCLIP: Instantiation of Algorithm 1 through embedding similarity.

Require: Offline dataset $\mathcal{D}$.

- 1: Initialize embedding models, $\mathbf{w}$ and $\mathbf{u}$, with the CLIP encoder.
- 2: Initialize policy parameters θ .

- 3: **procedure** OPTIONSPECIFICATION($\mathcal{O}, \mathcal{D}$)

- 4: Fine-tune multi-modal embeddings, $\mathbf{w}$ and $\mathbf{u}$, through Eq. 96 and Eq. 97 on $\mathcal{D}$.

- 5: Define option objectives as $\Omega = \{r^g\}$ where $r^g(s_{t-16:t}) = \mathbf{w}(s_{t-16:t})^\top \mathbf{u}(g)$.

- 6: **return** Ω

- 7: **end procedure**

- 8: **procedure** OPTIONLEARNING($\mathcal{O}, \mathcal{D}, \Omega$)

- 9: Construct options $\mathcal{O}$:

- 10: Update $\pi_\theta(\cdot|s, g)$ using transitions sampled from $\mathcal{D}$ to maximize $r^g \in \Omega$.

- 11: Time-induced termination β after fixed H_o timesteps.

- 12: $\forall s, \mathcal{J}(s) = 1$

- 13: **return** $\mathcal{O}$

- 14: **end procedure**

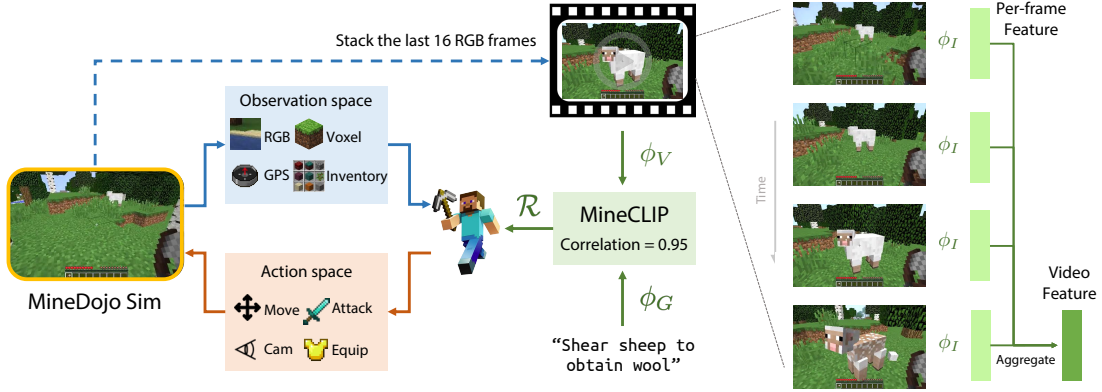


Figure 11: Illustration of the method of embedding similarity for defining option reward functions. Visual observations and language goal descriptions are converted into embeddings, and their similarity (e.g., via MineCLIP) is used to define reward functions for goal-conditioned policies. In this example, the agent is rewarded for successfully performing sheep shearing. Figure taken from Fan et al. (2022).

Fan et al. (2022) instantiate this idea in the open-ended Minecraft game (Johnson et al., 2016; Kanervisto et al., 2022). To do so, they introduce the MineDojo framework shown in Figure 11. The authors collect a large dataset of Minecraft gameplay for training a reward function that would map textual goals and a sequence of observations to a scalar value indicating their similarity. We present the algorithm in Algorithm 13. The language goal is encoded through the pretrained CLIP encoder (Radford et al., 2021) whereas the video encoder is composed of an image encoder and a temporal aggregator that accumulates 16

consecutive frames from the video. This leads to the following non-Markovian reward,

$$r^g(s_{t-16:t}) = \mathbf{w}(s_{t-16:t})^\top \mathbf{u}(g). \quad (98)$$

The authors train their reward model, called MineCLIP, on the aforementioned dataset using the same losses as in Equation 96 and 97. This resulting reward function excels at capturing correct behaviour on a wide collection of tasks, such as “Combat zombie”. Lifshitz et al. (2023) build on this work to obtain an instruction-following agent in Minecraft, where language instructions represent goals.

CLIP-based methods have also been applied to robotics. Xiao et al. (2022) fine-tune the CLIP model on a small dataset of robotic tasks and then utilize the model to label, using a set of predefined annotations, a much larger dataset of unlabeled observations. Using this larger dataset, the authors then train language-conditioned policies to achieve goals through imitation learning. Further improving the sample efficiency of embedding-based methods, Palo et al. (2023) show the possibility of efficiently fine-tuning the same CLIP model on as little as 1000 data points. Avoiding the costly operation of fine-tuning large pretrained models, Cui et al. (2022) investigate the prospect of using the CLIP model in a zero-shot fashion for defining goal-conditioned policies, obtaining good results on robotics tasks. Similarly, Rocamonde et al. (2024) leverage a fixed pretrained CLIP model and study the scaling effect of such models on the resulting RL performance.

To tackle long-horizon tasks, methods like JARVIS-1 (Wang et al., 2023c) propose hierarchical frameworks that employ LLMs as high-level planners to sequence options that were obtained by the methods described in this section, that is, embedding-based options. The planner selects options by querying a multimodal memory of past successful plans, generates short-horizon subgoals, and validates them via a “self-check” simulation before dispatching them to a low-level controller for execution. Execution terminates upon goal completion, when a task-specific step limit expires, or prematurely if environment feedback signals an error. Such execution failures immediately prompt the high-level planner to diagnose the underlying issue and generate a newly sequenced plan.

6.1.1 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

The main question raised by the embedding similarity-based method in Algorithm 13 is where the embedding models come from. As we have covered in the previous paragraph, this can be achieved either through large datasets from scratch or fine-tuning existing models on smaller datasets. Termination is typically independent of these vectors and defined through a time limit (see Design Principle 2, Section 3.2.2).

6.1.2 BENEFITS AND OPPORTUNITIES

Exploration. Methods building on embedding-based rewards empirically show improved exploration in complex tasks. In particular, in open-ended environments such as Minecraft, the dense nature of the reward functions obtained from embedding similarity significantly helps with exploration, leading to sophisticated behaviour (Fan et al., 2022; Lifshitz et al., 2023). The dense nature of such reward functions is also particularly useful for approaches studying the challenge of robotics (Xiao et al., 2022; Fu et al., 2024) and web navigation (Baumli et al., 2023). Du et al. (2023c) investigate how guiding exploration with an LLM

during a pretraining phase can help an agent’s downstream performance. To do so, the authors introduce the idea of restricting the reward function through a similar threshold,

$$r^{g,T}(s_t, a_t, s_{t+1}) = \begin{cases} r^g(s_t, a_t, s_{t+1}) & \text{if } > T, \\ 0 & \text{otherwise,} \end{cases} \quad (99)$$

where T , the threshold, is a hyperparameter. This reduces the noise of possibly imperfect embeddings used to define the reward function, further improving exploration.

Transfer. Another important benefit from LLM-based approaches to skill discovery stems from the compositional nature of language, which easily allows for specifying a variety of goals. For example, Du et al. (2023a) study how pretraining the agent on self-generated goals, where good behaviour is rewarded by the embeddings of an LLM, can lead to improved downstream performance on a set of complex goals. To encourage reaching a diversity of goals that will transfer well, the authors additionally prompt the LLM to generate k goals and reward the agent on the goal with the greatest reward,

$$r^{g_{max}} = \max_{i=1\dots k} r^{g_i,T}(s_t, a_t, s_{t+1}). \quad (100)$$

Similar generalization to different language-conditioned goals is reported by Lifshitz et al. (2023). Instead of directly training with a goal-conditioned model, Mahmoudieh et al. (2022) efficiently train a discrete set of smaller policies, used as a basis of behaviour. This is then distilled into a single language-conditioned neural network, which can better generalize on a larger spectrum of behaviours than the basis.

Opportunities for Research.

- *Understanding the trade-offs of different embeddings.* An important question when working with embedding similarity measures is with respect to the origin of the embeddings themselves. Most of the presented papers rely on CLIP, but other embeddings have been used, such as the Bidirectional Encoder Representations (BERT) embeddings (Devlin et al., 2019) and the Reusable Representations for Robot Manipulation (R3M) (Adeniji et al., 2023), which is pretrained on the Ego4D dataset (Grauman et al., 2022) through a combination of contrastive and video-language alignment losses. When considering a wide range of tasks, it is not clear which model shows greater performance, or is more amenable to fine-tuning.
- *Expanding beyond text-image similarity.* Most works compute the similarity between a language goal instruction and the current observation. Sontakke et al. (2023) instead compute the similarity between an agent attempting to reach a goal and a demonstration of such a successful behaviour. Moreover, contrary to most works, the authors compute the reward at the trajectory level, that is, the reward is only given at the end of an interaction. The authors show that their approach can be applied even in the case where the demonstration is done by a human physically completing the task, rather than teleoperating a robot, which presents greater opportunities for generalization.

6.2 Providing Feedback

Leveraging the embeddings of foundation models to measure the similarity between a desired goal and the current state places significant emphasis on the quality of the embeddings themselves. One way to avoid this is by considering the auto-regressive nature of LLMs, which allows for chain-of-thought (Wei et al., 2022) and in-context learning (Brown et al., 2020). Such capabilities can be particularly useful to define option reward functions. This can be done by taking as input a state, or a trajectory, as well as a goal description, and using LLMs to output a scalar feedback, representing the degree of success with respect to the goal. Alternatively, preferences can be elicited from an LLM over pairs of states that are then converted into a reward model through preference-based learning (Wirth et al., 2017), which can then be used to learn goal-conditioned low-level policies.

DIRECT REWARD

One of the most straightforward way to construct a reward function using foundation models is to employ them as direct evaluators of task success. This approach leverages the broad semantic understanding of large language models (LLMs) and vision-language models (VLMs) to autonomously verify whether an agent has completed a specified goal, bypassing the need to learn reward functions.

Formally, this paradigm models the reward as a direct mapping from an environment state $s \in \mathcal{S}$ and a natural language goal $g \in \mathcal{G}$ to a scalar success metric $y \in \mathcal{Y}$. For instance, to evaluate a binary success measure, Du et al. (2023a) pass a sequence of visual observations alongside a query such as “Did the agent successfully place the cactus left of the sofa?” to a multimodal model (Alayrac et al., 2022). The subgoal reward is then directly defined by this binary output $y \in \{0, 1\}$:

$$r^g(s) = y = \text{LLM}(s, g). \quad (101)$$

Such reward functions are evaluated on a diversity of domains: embodied simulations (Abramson et al., 2021), robotic manipulation with a 6DoF device, and human interactions in the Ego4D dataset (Grauman et al., 2022). To obtain accurate success measures, the authors have to initially fine-tune the model on a large dataset of expert interactions. Instead of costly model updates, Kwon et al. (2023) propose to replace weight updates with few-shot in-context examples, building on improved learning capabilities in the employed LLM. Pan et al. (2024) show that measures of success obtained from a multimodal LLM have high agreement (up to 92.9%) with oracle evaluators. Such results are reported on WebArena (Zhou et al., 2024) and Android-in-the-Wild (Rawles et al., 2023) benchmarks. Leveraging the strong performance of LLMs as direct reward modelers, Bai et al. (2024) successfully train robust RL policies on a variety of goals derived from changing web interfaces.

ELICITING PREFERENCES

When an LLM’s direct output serves as the reward signal, it may reliably detect success, but it often lacks the nuance required to properly gauge a specific state’s value compared to the full spectrum of alternatives. Instead, we can leverage reinforcement learning from AI feedback (Bai et al., 2022). By relying on preference-based learning (Wirth et al., 2017; Thomaz and Breazeal, 2006), we can obtain more refined valuations for each state that

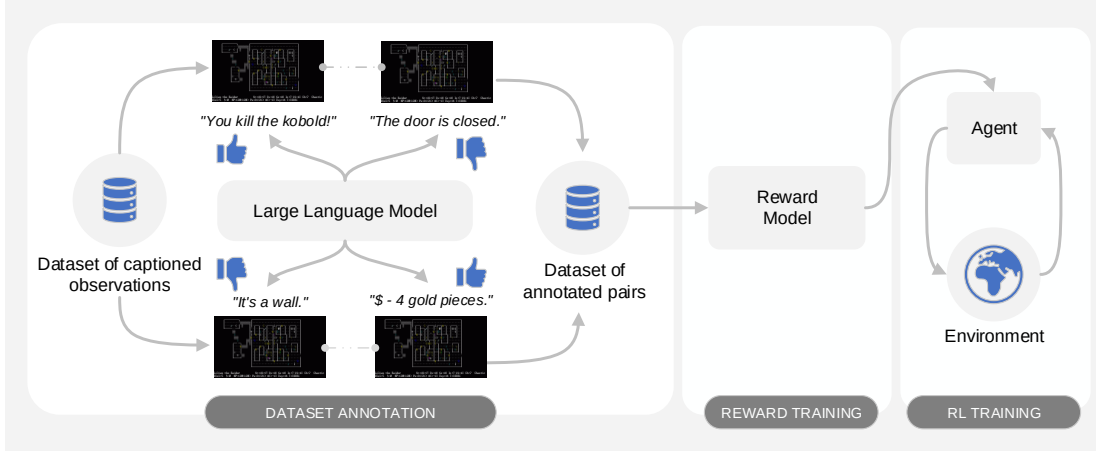


Figure 12: Learning option rewards from AI feedback proceeds in three phases. In the first phase, an LLM is conditioned on a behaviour description and queried for preferences over pairs of observations, which are stored with their preference labels within a dataset. In the second phase, the preferences are distilled into an observation-based scalar reward function. Finally, an agent is trained interactively with RL, receiving a scalar signal at every step through the reward function extracted from the preferences.

accurately reflect its relative merit within the much larger state space. Building on this idea, Klissarov et al. (2024) introduce the Motif algorithm, which leverages an LLM’s feedback to guide an agent acting in the open-ended NetHack environment (Küttler et al., 2020). Observations from the environment are presented to an LLM before querying, using chain-of-thought prompting, to provide a preference over which observation is more desirable for a certain goal. This process is shown in Figure 12.

Formally, the annotation function is given by $\text{LLM} : \mathcal{S} \times \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{F}$, where $\mathcal{S}$ is the space of states, where $\mathcal{G}$ is the space of goals defined through natural language, and $\mathcal{F} = \{1, 2, \emptyset\}$ is a space of preferences for either the first, the second, or none of the captions. These preferences are then distilled into a reward function through the Bradley-Terry model (Bradley and Terry, 1952) and given to an RL agent interacting with the environment,

$$\mathcal{L}(\nu) = -\mathbb{E}_{(s_1, s_2, g, y) \sim \mathcal{D}_{\text{pref}}} \left[\mathbb{1}[y = 1] \log P_\nu[s_1 \succ s_2 | g] + \mathbb{1}[y = 2] \log P_\nu[s_2 \succ s_1 | g] + \mathbb{1}[y = \emptyset] \log \left(\sqrt{P_\nu[s_1 \succ s_2 | g] \cdot P_\nu[s_2 \succ s_1 | g]} \right) \right], \quad (102)$$

where $P_\nu[s_a \succ s_b | g] = \frac{e^{r_\nu^g(s_a)}}{e^{r_\nu^g(s_a)} + e^{r_\nu^g(s_b)}}$ is the probability of preferring a state s_a to another s_b given a goal g ; r_ν^g is the reward defined with respect to the goal specified in the LLM’s prompt.

Through the process of comparing states to alternatives, eliciting LLM preferences, or receiving AI feedback, nuanced and fine-grained reward functions can be provided. Such reward functions can also be understood as process-based rewards (Uesato et al., 2023;

Algorithm 14 MaestroMotif: Instantiation of Algorithm 1 by providing feedback.

Require: Offline dataset $\mathcal{D}$ of interactions.

- 1: Initialize policy parameters θ and option reward parameters ν .
 - 2: **procedure** OPTIONSPECIFICATION($\mathcal{O}, \mathcal{D}$)
 - 3: Prompt the LLM to describe each option’s intended behaviour textually.
 - 4: Optimize Eq. 102 on $\mathcal{D}$ through binary cross-entropy, where preferences are queried w.r.t. the option’s language description.
 - 5: Define option objectives as $\Omega = \{r_\nu^o\}$.
 - 6: **return** Ω
 - 7: **end procedure**
 - 8: **procedure** OPTIONLEARNING($\mathcal{O}, \mathcal{D}, \Omega$)
 - 9: Construct options $\mathcal{O}$:
 - 10: Update $\pi_\theta(\cdot|s, o)$ through online interactions to maximize $r_\nu^o \in \Omega$.
 - 11: $\mathcal{I}(s)$ and $\beta(s)$ defined symbolically through LLM-generated code.
 - 12: **return** $\mathcal{O}$
 - 13: **end procedure**
-

Lightman et al., 2023). Klissarov et al. (2024) leverage this characteristic to learn a set of policies that exhibit a certain behaviour across time, such as preferring generally more cautious strategies when exploring. This is in contrast to the work on LLM as direct reward modelers, which typically define rewards for reaching goal states as binary success detectors (Du et al., 2023a). As illustrated in the MaestroMotif algorithm (Klissarov et al., 2025a), the flexibility offered by AI feedback is key in designing HRL agents capable of subtle behaviours and fast adaptation. Adding to the generality of AI feedback, Wang et al. (2024a) investigate the resulting policies across a range of continuous control domains using pixel observations and a multimodal LLM. Their findings show that reward functions generated through AI feedback yield more performant policies compared to embedding similarity approaches or methods that directly query the LLM for scalar rewards.

6.2.1 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

Algorithm 14 details the MaestroMotif method presented in Klissarov et al. (2025a). By leveraging preferences to define a reward function, the option objective can capture nuanced relationships between different states in the open-ended domain of NetHack. However, the authors did not use the same preference-based mechanism to define the initiation and termination functions. Instead, they opted for a symbolic representation through code, which is possibly simpler and more compute-efficient.

6.2.2 BENEFITS AND OPPORTUNITIES

Exploration. Klissarov et al. (2024) illustrate the potential of AI feedback-based rewards to significantly improve exploration on the complex open-ended world of NetHack. The obtained reward function is shown to be naturally dense and encodes a variety of important

milestones, such as unlocking doors or picking up items. The authors hypothesize that, by querying the model on thousands of pairs of observations from the environment, the LLM’s common sense reasoning and domain knowledge are distilled into a useful reward function.

Credit Assignment. Wang et al. (2024a) report that the reward obtained from preferences monotonically increases as the agent advances towards the goal, naturally assigning credit to states in between the starting state and the goal. Klissarov et al. (2025b) further study the dense nature of such reward functions and reveal a strong correlation with value functions obtained at the end of training. As such, value functions have been trained to propagate information through temporal difference learning (Sutton, 1988), the authors argue that this high correlation is another indication that the reward functions based on LLM feedback are useful for credit assignment. An equivalent perspective is that the resulting dense reward can be seen as a form of reward redistribution (Arjona-Medina et al., 2019; Hung et al., 2019; Klissarov and Precup, 2020; Ni et al., 2023), which is an established method for improving credit assignment.

Transfer. In MaestroMotif, Klissarov et al. (2025a) show how a set of semantically meaningful skills can be easily re-composed zero-shot to adapt to complex new tasks. Leveraging the code generation abilities of LLMs, they propose a neuro-symbolic approach where skill policies are neural networks trained by reinforcement learning, and the high-level policy is defined through code. The authors then use the in-context learning abilities of LLMs to re-compose the skills, significantly outperforming baselines that are trained specifically on each of the tasks. Their approach highlights how the compositional nature of language can be particularly helpful when combined with a set of linguistically-defined skills, leading to an easily promptable agent.

Opportunities for Research.

- *Simplifying the reward learning process.* Despite the strength of preference-based methods for crafting rewards through LLMs, they are more complex than directly querying for a reward signal. Is there a way to leverage the improved exploration and credit assignment without the additional complexity? Is an existing dataset of observations needed for eliciting useful preferences? Zheng et al. (2024) provide an initial answer to these questions by comparing different ways in which the LLM feedback is leveraged, for example, by using it as a label for a classification loss. Their results show surprisingly strong performance of some of these simpler baselines, even when querying the LLM with online interactions.

6.3 Reward as Code

Instead of relying on LLMs to evaluate good and bad behaviour from observations, it is possible to rely on their code generation abilities to write reward functions which can be used immediately to learn low-level goal-conditioned policies. In this line of work, a goal description is given to the LLM as input, as well as additional information from the environment,

$$\text{code}^g \sim \text{LLM}(g, \text{info}), \quad (103)$$

$$r^g(s) = \text{code}^g(s). \quad (104)$$

Algorithm 15 Instantiation of Algorithm 1 through reward as code.

```
1: Initialize policy parameters  $\theta$ .

2: procedure OPTIONSPECIFICATION( $\mathcal{O}, \mathcal{D}$ )
3:   Prompt an LLM to define the code implementing the reward function.
4:   Define option objectives as  $\Omega = \{r^g\}$  where  $r^g = \text{code}^g(s)$ .
5:   return  $\Omega$ 
6: end procedure

7: procedure OPTIONLEARNING( $\mathcal{O}, \mathcal{D}, \Omega$ )
8:   Construct options  $\mathcal{O}$ :
9:     Update  $\pi_\theta(\cdot|s, g)$  through online interactions to maximize  $r^g \in \Omega$ .
10:    Time-induced termination  $\beta$  after fixed  $H_o$  timesteps.
11:     $\forall s, \mathcal{J}(s) = 1$ 
12:   return  $\mathcal{O}$ 
13: end procedure
```

This additional information often constitutes important symbolic information, such as low-level features, that is used to define the code. This code is then executed alongside the environment simulation to generate a reward for every state s . Xie et al. (2024) explore the possibility of leveraging an LLM’s capacity to code reward functions for robotics tasks. The authors provide the LLM with additional information in the form of a symbolic representation of the environment (e.g., Python classes describing each object and methods to access specific information about it). Furthermore, the authors provide the LLM with helpful functions from different packages (such as quaternion computation in NumPy) to be used for reward generation. Finally, their algorithms also allow for integrating human feedback. Such a framework is shown in Figure 13. Yu et al. (2023) similarly investigate how LLMs can generate reward functions for learning robotics skills. In their approach, an LLM takes as input a detailed language description of a goal and instantiates a set of reward functions.

Another notable work is that of Ma et al. (2024), which presents Evolution-driven Universal REward Kit for Agent (EUREKA). They provide a task description to the LLM, such as “make the pen spin to a target orientation”, and proceed to do an evolutionary search on the reward function space. This process is supported through additional context given to the LLM in the form of selected parts of the source code of the environment. For each candidate reward function generated by the LLM, a full training run is executed via massively distributed RL experiments within Isaac Gym (Makovychuk et al., 2021). The most promising reward function candidates are then retained and given to the LLM together with the learning statistics, such that the model performs in-context learning and suggests a new batch of candidates.

6.3.1 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

Algorithm 15 relies entirely on the coding ability of LLMs. Given that code that represents rewards for simulated robotics is present on the internet, it is a reasonable assumption to expect LLMs to be able to output bespoke variations for different behaviours.

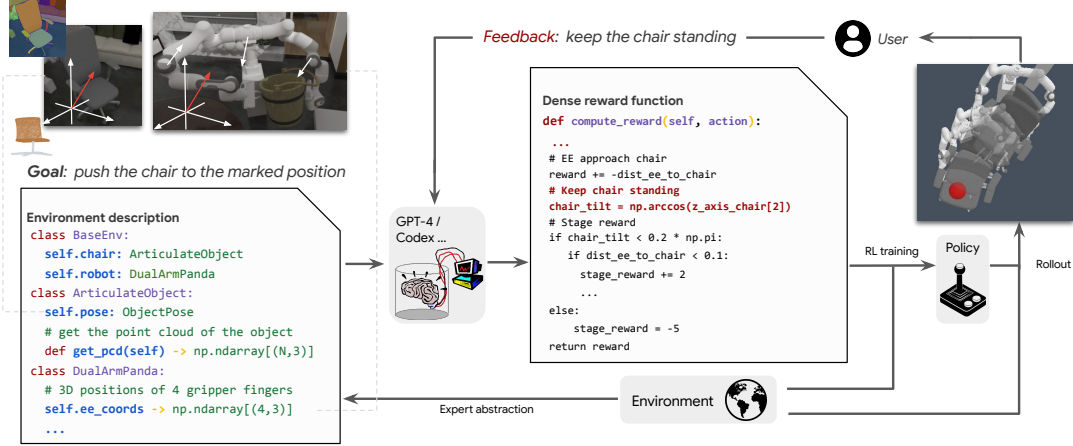


Figure 13: Defining reward functions as code requires access to a symbolic representation of the environment. This is done through an *expert abstraction* function that represents the environment as a hierarchy of Pythonic classes. The *user instruction* describes, in natural language, the goal to be achieved. The agent then interacts with the environment to maximize this symbolic reward function. It is also possible to include *user feedback* that summarizes the failure modes of the current reward code. Figure taken from Xie et al. (2024).

6.3.2 BENEFITS AND OPPORTUNITIES

Transfer. The ability to efficiently generate reward functions, without human supervision, is particularly important for transfer. For example, Ma et al. (2024) achieve super-human level reward design for complex robotics skills across a variety of embodiments. In the domain of Minecraft, Li et al. (2024) show how reward functions as code can be used to solve a variety of long-horizon goals given access to the symbolic features from the environment.

Opportunities for Research.

- *Going beyond symbolic representations.* Generating a reward function as code is a powerful paradigm: it avoids the need to query the LLM during the RL phase and does not require learning a parametric reward model. However, by definition, such an approach requires access to symbolic features from the domain of interest, which can be limiting if this involves real-world interactions with humans. Venuto et al. (2024) propose to query the LLM to craft its own symbolic representation from high-dimensional observations, similar to the work by Palo and Johns (2024). These representations are then used to define reward functions in code. However, their approach requires access to expert demonstrations, which future work could alleviate.

6.4 Directly Modeling the Policy

So far, we have covered methods that leverage foundation models to define goal reward functions through a variety of strategies, such that goal-conditioned policies can be obtained by maximizing the reward functions. Alternatively, there exists a line of work that uses LLMs to directly model the policy itself, where goals are defined through prompts and conditioning the LLM on them, effectively serving as goal-conditioned policies. In this

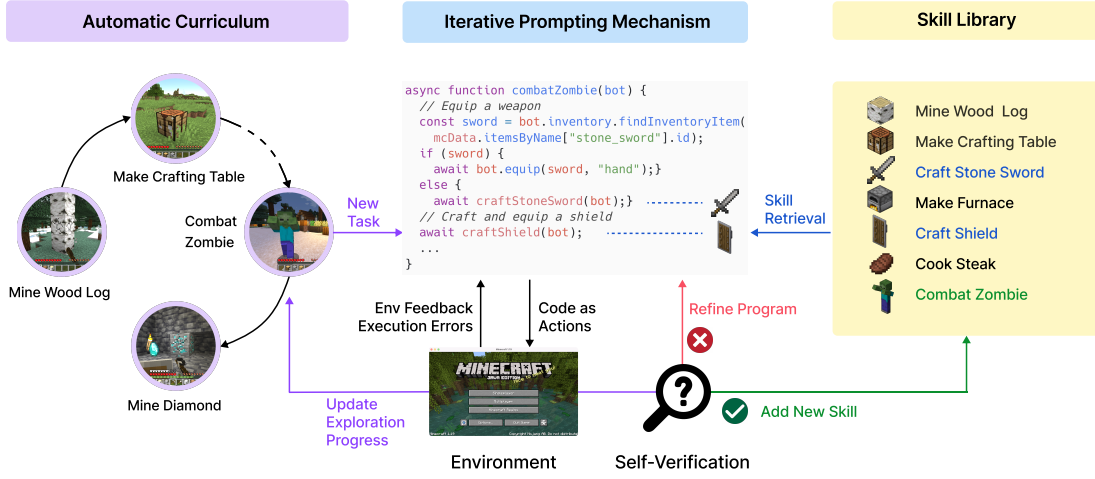


Figure 14: An LLM is conditioned on a goal description and generates snippets of code which instantiate skill policies. When interacting in multimodal environments, such as Minecraft, a bridge between this symbolic skill policy representation and the high-dimensional nature of the environment has to be present. Under such a setting, an LLM can act as an HRL agent, efficiently achieving complex goals. Figure taken from Wang et al. (2023a).

setting, the LLM is oftentimes updated through in-context learning (Wei et al., 2022) to obtain policy improvements, bypassing the need for performing parameter updates, which can be costly and time-consuming. Building on the code generation capabilities of LLMs, Liang et al. (2023) propose to define robotic skills as policies in the form of Python code,

$$\text{code}^g \sim \text{LLM}(g), \quad (105)$$

$$a \sim \text{code}^g(s), \quad (106)$$

where the goal-conditioned code^g acts as a the goal-conditioned policy $\pi(a|s, g)$. They show that the LLM can re-compose calls to an API such that a new code policy achieves a specific goal. In particular, they propose hierarchical code-generation that recursively defines undefined functions from existing functions, leading to strong performance on robotics tasks. Kwon et al. (2024) extend this work, removing assumptions such as providing in-context examples or requiring the LLM to predict end-effector poses. In the complex open-ended environment of Minecraft, Wang et al. (2023a) propose Voyager (Figure 14), a method leveraging an LLM to continually expand a library of skills. Such skills are crafted by prompting the LLM to define specific behaviours in code, building on an existing JavaScript API (PrismarineJS, 2013) that allows for grounding the generated code in the multimodality of Minecraft. Voyager further uses ideas of auto-curriculum and self-reflection to update the set of skills, learn new ones, or define their composition for a given task.

Another line of work directly queries the LLM for actions by giving as context the natural language description of a goal and the current state,

$$a \sim \text{LLM}(s, g). \quad (107)$$

In these settings, LLMs output the low-level actions in an environment, effectively as a goal-conditioned policy. This particular instantiation highlights that LLMs are already

Algorithm 16 Instantiation of Algorithm 1 by directly querying the LLM.

```

1: procedure OPTIONSPECIFICATION( $\mathcal{O}, \mathcal{D}$ )
2:   Prompt an LLM to specify the goal description  $g$  in natural language or code.
3:   Define option objectives as  $\Omega = \{g\}$ .
4:   return  $\Omega$ 
5: end procedure

6: procedure OPTIONLEARNING( $\mathcal{O}, \mathcal{D}, \Omega$ )
7:   Construct options  $\mathcal{O}$ :
8:      $\pi_o(a \mid s) = \text{LLM}(a \mid s, g)$ 
9:      $\forall s, \mathcal{J}(s) = 1$ 
10:   Terminate when the LLM outputs the end of generation token.
11:   return  $\mathcal{O}$ 
12: end procedure

```

particularly effective HRL agents that can be conditioned on goal descriptions. The current focus of such models is on computer-based tasks (Anthropic, 2024; OpenAI, 2025). Despite the appeal of generalizing zero-shot to new language instructions, current LLMs are still quite limited in successfully performing long-horizon tasks by directly selecting low-level actions (OpenAI, 2024; Zhou et al., 2024).

6.4.1 RELATIONSHIP TO THE ALGORITHMIC TEMPLATE

In Algorithm 16 the option objectives take an abstract form: language descriptions. Given the algorithmic simplicity and the modular nature of language, it is currently the most widespread method for defining skills in LLM-based AI agents.

6.4.2 BENEFITS AND OPPORTUNITIES

Exploration. By relying on an LLM’s common sense, prior knowledge, and possible API libraries, researchers have shown that agents explore their environment significantly better. By directly modeling the policy, it is possible to condition the LLM on a wider variety of goals and find well-performing policies for a subset of easier goals. This allows for making progress on very hard exploration problems by breaking the task into achievable milestones. Examples include collecting diamonds in Minecraft (Wang et al., 2023a) or intricate web navigation tasks (Zhou et al., 2024). By conditioning an LLM on language and directly outputting a sequence of actions to achieve tasks, agents achieve tasks that would be extremely difficult, or even impossible, to learn by maximizing a reward function.

Transfer. Directly acting with an LLM greatly simplifies how users can leverage the compositional nature of language. For example, the same LLM can be directly conditioned on a variety of computer interaction tasks and achieve them zero-shot (Anthropic, 2024). Alternatively, a library of skills can be re-composed through in-context learning to craft new skills (Wang et al., 2023a).

Opportunities for Research.

- *Lifting restrictions on the action space and action frequency.* The prospect of directly generating a wide spectrum of behaviours simply by querying a large pretrained model is particularly appealing. It essentially encompasses the fundamental promise of HRL for fast adaptation thanks to the compositional nature of language. However, it also poses interesting challenges. For example, would such a model be restricted to a certain action space, or is there a way to efficiently adapt to a variety of embodiments? Are there limitations in terms of action frequencies? The domain of computer navigation is especially promising as grounding an LLM in the action space of computers would allow a model to achieve many economically useful tasks. However, the same model could not be used to control an embodied robot, unless fine-tuning is performed, which for large models is costly. A varying action space also raises the necessity to co-fine-tune the model to avoid catastrophic forgetting (Brohan et al., 2023).

7. Using Discovered Options

In the previous sections, we presented a variety of approaches addressing the option discovery problem. This naturally leads to the question: once an agent has discovered a set of options, how does it use them to select primitive actions in the environment? We refer to the mechanism by which options translate into action selection as the *execution model*. In all execution models, the agent pays the computational cost of selecting a primitive action at every timestep—the same cost incurred in flat RL. HRL adds the computational cost of deliberating over options, which might involve a forward pass through a large network, a planning computation using option models, or chain-of-thought reasoning in an LLM. This added cost can pay for itself; for example, planning with options can require far fewer value-iteration backups to reach a near-optimal solution (Mann et al., 2015). An execution model is fully specified by two choices: *when* the agent deliberates over options, and *how* it uses options to select primitive actions.

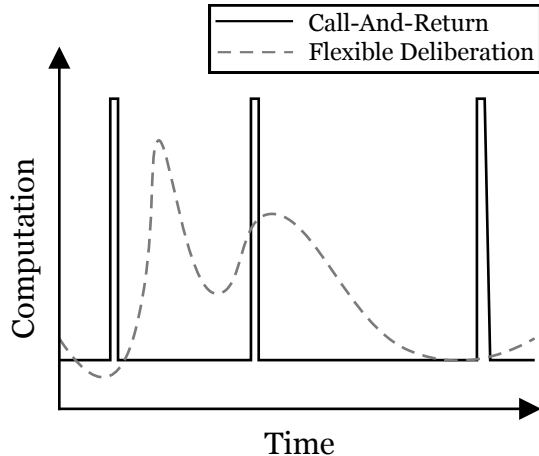


Figure 15: Depiction of the distribution of computation over time for the standard call-and-return model of execution. High-level decisions incur a greater computation cost than low-level ones, producing the spikes that characterize call-and-return. We also present a hypothetical model that distributes computation more flexibly across timesteps.

7.1 When to Deliberate

At one extreme, deliberation occurs only at sparse decision points (e.g., upon option termination under the call-and-return execution model); at the other, the agent reconsiders its choice at every timestep. These endpoints trade off deliberation cost against responsiveness; we illustrate this spectrum in Figure 15. One way to think about this tradeoff is through

the lens of dual-process theories of cognition (Kahneman, 2011): selecting among options resembles slow, deliberate “System 2” reasoning—for example, model-based planning using option models—while acting according to the current option’s policy resembles fast, habitual “System 1” action selection. The less frequently the agent engages System 2, the lower the per-step computation overhead, but the greater the risk of persisting with a suboptimal option.

7.1.1 CALL-AND-RETURN

In the call-and-return model, the agent’s high-level policy selects a single option at each *decision point*—a timestep at which no option is currently executing, i.e., the start of the episode or the termination of the previous option. The selected option then governs action selection until it terminates, at which point the high-level policy selects again. This is the most widely used execution model and was used to describe HRL in Figure 4. Call-and-return concentrates the computational cost of deliberation at option boundaries: the agent amortizes the cost of a single deliberation over the option’s duration, so the longer the option persists, the lower the per-step overhead (Wen et al., 2020).

Because options execute for variable durations, the time between consecutive high-level decisions is a random variable. The decision process at the option level is therefore a semi-Markov decision process (SMDP, see Section 1 for details) (Bradtke and Duff, 1994). In practice, this requires only minor modifications to standard algorithms. In the model-free case, the experience tuple for updating the high-level policy becomes $(s_t, o_t, \sum_{k=0}^{\tau} \gamma^k r_{t+k}, s_{t+\tau})$, where τ is the duration of option o_t . In the model-based case, the reward model predicts expected discounted cumulative reward during execution, and the transition model predicts the discounted probability of terminating in $s_{t+\tau}$ (Sutton et al., 1999b).

A potential limitation of call-and-return is **commitment**: the agent cannot change course mid-option, even if the option’s behaviour becomes suboptimal given newly visited states. The agent must wait for the option to terminate before it can deliberate again.

7.1.2 INTERRUPTIONS

The commitment problem motivates a natural extension: allowing the agent to interrupt a running option before its natural termination and select a new one (Sutton et al., 1999b). Rather than waiting for termination, the agent periodically checks whether the current option should be replaced, introducing a parameter k that controls how often this check occurs. When k equals the full option duration, this recovers call-and-return; when $k = 1$, the agent reconsiders at every timestep. The parameter k thus provides a knob that interpolates between the two extremes of the computation cost spectrum.

One concrete interruption criterion analyzed by Sutton et al. (1999b) is to interrupt option o in state s whenever $Q(s, o) < V(s)$ —that is, whenever the advantage of the current option is negative and the agent could do better by switching. Under this criterion, and when option policies and termination conditions are fixed, the interrupting execution model yields a value function at least as great as call-and-return for any policy over options.

The value of k also determines the Markov properties of the resulting process. At $k = 1$, the process over the augmented state (s, o) is Markov: the action and switching probability depend only on the current state and option, with no dependence on how long the option

has been executing. For $k > 1$, the process becomes k -Markov in (s, o) —the agent must also know how many steps remain until the next interruption check, which can be captured by augmenting the state with a counter.

When options are being *learned* rather than held fixed, unrestricted interruption can cause the hierarchy to degenerate: the agent may learn to terminate every option after a single timestep, collapsing to flat control (Bacon et al., 2017).

7.2 How to Select Actions

Given a deliberation schedule, the agent must also decide how to use its options to produce a primitive action at each timestep. The simplest approach is to follow the policy of a single active option. An alternative is to combine the policies or value functions of multiple options, selecting each action by drawing on the collective knowledge of the option set.

7.2.1 SINGLE OPTION

The most straightforward approach is to follow the policy of the currently active option: at each timestep, the agent executes whatever action that option prescribes. This is the standard assumption in call-and-return and interruptions as described above.

7.2.2 OPTION COMPOSITION

An alternative is to combine the value functions of *multiple* options to select each action. Generalized policy improvement (GPI) (Barreto et al., 2017, 2018, 2020) is a concrete instantiation. Given a set of policies with associated action-value functions $Q_1, \dots, Q_n$, GPI selects $a = \arg \max_a \max_i Q_i(s, a)$. This provides a guarantee: the resulting policy performs at least as well as the best individual policy in every state. Note that the Q_i here are standard action-value functions for flat policies—there is no initiation set, termination condition, or notion of commitment; GPI queries each policy’s value function and cherry-picks the best action at each timestep. Other approaches combine option value functions directly (Todorov, 2009a; Hunt et al., 2018; Haarnoja et al., 2018), for instance by taking convex combinations.

Composing the two choices. The choice of action selection mechanism is orthogonal to the choice of deliberation frequency, and interesting execution models arise from combining them. The *option keyboard* (Barreto et al., 2019) is a clear illustration. It defines composite options whose internal policy uses GPI-style maximization over multiple base-option value functions at every timestep, while a high-level call-and-return policy selects *which* composition to pursue. Concretely, suppose the agent has n base options, each with a policy π_i , an action-value function Q_i , and a *cumulant* e_i —an option-specific reward signal capturing what the option is trying to achieve (Sutton et al., 2011). A new composite option is specified by a weight vector $w \in \mathbb{R}^n$ that linearly combines the base cumulants into $e = \sum_i w_i e_i$. Let $Q_i^e(s, a)$ denote the value of taking action a in state s and thereafter following base policy π_i , evaluated under the combined cumulant e . At each timestep, the composite option selects

$$a = \arg \max_a \max_i Q_i^e(s, a), \quad (108)$$

continuing until a termination condition is met. The high-level policy then chooses a new weight vector and the process repeats. Within the composite option, multiple value functions

are queried at every timestep (composition); between options, the high-level policy deliberates over weight vectors (call-and-return). At the other extreme, combining composition with interruptions at $k = 1$ recovers standard GPI—the agent composes value functions at every timestep, thereby precluding the need for a high-level policy, with no temporal commitment and a fully Markov process. The option keyboard has recently been instantiated with options derived from the graph Laplacian (Chandrasekar and Machado, 2026), illustrating how discovery methods from Section 4.2 supply the base options that composition mechanisms then combine; we return to this in Section 8.2.

The execution models above define how an agent converts its options into actual decisions. They leave open the question of how the agent learns which options to invoke; we turn to this question next.

7.3 Learning High-level Policies

The agent’s high-level policy, $\mu(o|s)$, is responsible for selecting an option. We present different approaches to learning this quantity by separating methods into three categories: model-free approaches, model-based approaches, and approaches that rely on in-context learning using foundation models.

7.3.1 MODEL-FREE APPROACHES

Usual model-free RL methods (like Q-learning) can, with slight modifications, be used to learn a policy that selects options $\mu(o|s)$. These modifications simply involve discounting rewards obtained during option execution appropriately and using the state at the end of option execution as the next environment state, i.e., the experience tuple used to update the agent is $(s_t, o_t, \sum_{k=0}^{\tau} \gamma^k r_{t+k}, s_{t+\tau})$, where τ is the duration of execution of option o_t (Bradtke and Duff, 1994). This approach, while simple, treats option execution as a black-box. When the chosen option is Markov, meaning that its duration τ can be written purely as a function of state (and not time), then intra-option learning can be used for improved sample-efficiency. As long as states observed during option execution are inside the option’s initiation set, then the corresponding transitions can be used to update $\mu(o|s)$ (Sutton et al., 1998). Specifically, an SMDP transition $(s_t, o_t, \sum_{k=0}^{\tau} \gamma^k r_{t+k}, s_{t+\tau})$ can be decomposed into up to τ transitions of the kind $(s_i, o_t, \sum_{k=0}^{\tau} \gamma^k r_{i+k}, s_{i+\tau})$ for all i such that $s_i \in \mathcal{I}_o$. Bacon (2018) later generalize these insights to policy gradient methods by proposing the option gradient theorem.

Bandits that maximize learning progress. A popular model-free approach is to treat the high-level policy as a contextual bandit (which can be thought of as an MDP with $\gamma = 0$). The reward function for the bandit is designed to carefully trade off various objectives. For example, when the extrinsic reward is dense and informative, the bandit simply chooses the option expected to maximize the reward (Schaul et al., 2019). When the reward function is sparse or deceptive, then a measure of *learning progress* (LP) is often added to the extrinsic reward; the idea is that the agent should pick options that (in addition to greedily maximizing reward) would also improve its knowledge of the environment and its own competence in the environment (Colas et al., 2022). Although measuring LP itself is intractable, proxies are used in practice. Competence progress (Oudeyer and Kaplan, 2007; Stout and Barto, 2010) prioritizes skills whose capabilities change the most with time—these skills represent subgoals of intermediate difficulty (Florensa et al., 2018). Count-based bonuses prioritize

options that lead to high novelty (Bagaria and Schaul, 2023; Badia et al., 2020b,a), and density-based approaches (Pong et al., 2020) attempt to maintain a high entropy distribution for option selection from different states (Pitis et al., 2020).

7.3.2 MODEL-BASED APPROACHES

In the model-free case, the high-level policy $\mu(o|s)$ is a learned mapping from states to options. In the model-based case, this mapping is computed on the fly: the agent learns transition and reward models for its options, uses those models to plan, and the plan itself determines which option to execute next. In some model-based approaches, options are used only inside the planner—to propagate value over long horizons—and are never executed in the environment at all. The appeal of this approach is that options afford temporally-extended models of the world, which enable longer-horizon planning. When the agent instead learns single-timestep models, it must roll them out over a long horizon, and model-prediction errors compound over time (Talvitie, 2017; Janner et al., 2019), leading to potentially massive errors in value approximation (Kearns and Singh, 2002). Options reduce this compounding by requiring fewer model applications to cover the same horizon (Mann et al., 2015)—but they do not eliminate it entirely, since option models are themselves learned via bootstrapping and can accumulate bias at long timescales (Farebrother et al., 2025, 2026).

Regardless of the type of option model, several shared concerns arise during learning. The agent’s interaction data can be used to learn option models either on-policy, updating the model for an option after it is executed (Sutton et al., 1999b), or off-policy, using intra-option learning (Sutton et al., 1998) to simultaneously learn about many options from the data collected at every timestep. Some methods learn in the agent’s raw observation space, which must contend with high-dimensional inputs and outputs (Nair and Finn, 2020); focusing on local, subgoal-conditioned models can ease this burden (Lo et al., 2024). Others operate in an abstract state space, which introduces the challenge of *drift*: option models must rapidly adapt to changes in the evolving abstract representation.

The most basic use of an option model is to evaluate a single option: predict what will happen if the agent executes option o from state s , and use that prediction to estimate the option’s value. But the real power of option models emerges when the agent plans over a *sequence* of options—evaluating the expected return of executing option 1, then option 2 from wherever option 1 terminates, and so on. To plan over such a sequence, the agent must represent the *handoff* between consecutive options: what does the world look like when one option terminates and the next begins? The choice of how to represent this handoff determines both what kind of sequential composition is possible and what planning algorithms can be used. We distinguish three families of approaches based on the information they retain about the boundary state.

(a) *Full distribution over boundary states.* The most expressive representation is to model the complete distribution over states that an option might terminate in. Silver and Ciosek (2012) showed that option models have a natural algebraic structure: an option model can be represented as a matrix bundling transition probabilities and rewards, and composing two options sequentially—executing one, then the other—corresponds to simple algebraic manipulations of their matrices. Option models are therefore closed under composition:

chaining two yields another option model of the same form, enabling recursive multi-level planning. However, representing full transition matrices is only tractable in finite state spaces. To extend compositional planning to continuous state spaces, recent work learns full generative models of the *successor measure*—the discounted distribution over future states induced by following a policy (Dayan, 1993). The discount factor γ can be interpreted as a continuation probability, making the option’s lifetime geometrically distributed; because the geometric distribution is both memoryless and closed under composition, chaining successor measures at different discount factors produces another successor measure of the same form. This observation underlies the *geometric horizon model* (GHM), a generative model of the successor measure conditioned on a policy and discount factor (Janner et al., 2020). Thakoor et al. (2022) showed how the successor measures of individual policies compose—via sampling—into the successor measure of a *geometric switching policy*, a sequence of policies where each executes for a geometrically distributed duration. Farebrother et al. (2026) generalized this to variable switching probabilities, allowing each policy in the sequence to persist for a different duration, and introduced a horizon-consistency training objective that anchors long-horizon predictions to shorter-horizon ones, reducing bias at long timescales. The strength of this family is unbiased composition via sampling; the limitation is that sampling-based planning (e.g., random shooting) does not scale as gracefully to long combinatorial plans as the symbolic methods discussed in (c).

(b) *Expected boundary state.* A simpler alternative is to predict only the expected terminal state and reward—an *expectation model* (Wan et al., 2019). When the value function is linear in the state representation, the expected state is sufficient: $\mathbb{E}[V(s')] = V(\mathbb{E}[s'])$, so no information is lost (Schoknecht, 2002; Parr et al., 2008; Sutton et al., 2008; Sorg and Singh, 2010). Expectation models are also efficient to learn, requiring only the solution of a system of linear equations (Sutton et al., 2023), which is especially useful when state representations evolve over time. However, when the agent chains multiple options, it must feed the predicted terminal state of one option into the model of the next. Under nonlinearity, this computes $V(\mathbb{E}[s'])$ rather than $\mathbb{E}[V(s')]$; by Jensen’s inequality, the resulting bias compounds with each link in the chain. Expectation models are therefore well-suited to evaluating individual options or short sequences under linearity, but less so to planning over long compositions in general settings.

(c) *Abstract boundary state.* A third approach, developed under the *skills-to-symbols* framework (Konidaris et al., 2018), is to compress the boundary representation through state abstraction. The resulting option models predict transitions between *abstract* states rather than in the original observation space (Li et al., 2006). When options drive all relevant state variables to a small range of values, the abstraction takes the form of a *graph*: nodes correspond to abstract states and edges to options, with an edge between two nodes when one option terminates in a state where another can successfully begin (Konidaris et al., 2018; Bagaria et al., 2021a, 2025). Options that do not control all state variables—setting some while leaving others unchanged—unlock a more structured representation: a type of factored MDP (Boutilier et al., 2000) described using Planning Domain Definition Language (PDDL) (McDermott et al., 1998). The advantage of generating a PDDL description is that it can be solved by off-the-shelf classical planners, even for long-horizon, combinatorial problems that would be intractable for sampling-based methods. The cost is that the abstraction is lossy: the agent discards information about where exactly within an abstract state it will

end up, which can lead to suboptimal plans (Abel et al., 2020). Rodriguez-Sanchez and Konidaris (2024) propose continuous abstract state representations that relax the discrete assumption while providing formal bounds on the resulting value loss, quantifying exactly how much reward the agent foregoes by planning in the abstract space rather than the original one. James et al. (2020) address a complementary challenge—transferability—by learning abstract representations expressed from the agent’s own perspective, which reduce sample requirements across sequences of related tasks (Konidaris and Barto, 2007).

In summary, the three families of boundary representations—distributional, expected, and abstract—trade off the richness of the handoff information against the tractability of the resulting planner. Distributional models enable unbiased composition via sampling, expectation models offer efficiency under linearity but degrade under chaining, and abstract models sacrifice distributional detail for access to powerful combinatorial planning algorithms. The choice depends on the structure of the problem: whether the state space is continuous or discrete, whether options have tight or diffuse effects, and whether the planning horizon demands combinatorial reasoning.

7.3.3 IN-CONTEXT LEARNING

If the high-level policy over options is implemented using a foundation model, e.g., an LLM, in-context learning can be used to learn the high-level policy, $\mu(o|s)$. This can be done by having the LLM output Python code that implements skill-selection logic (Wang et al., 2024c; Klissarov et al., 2024), or to output formal plans described using PDDL (Silver et al., 2024). Such policies can then be updated by providing execution traces as context to the LLMs and asking for code refinements. It is also possible to directly deploy the LLM in the environment to select skills at every high-level decision point (Ahn et al., 2022). Since such approaches do not require gradient updates, they potentially offer faster adaptation. However, the nature of in-context learning is currently not well understood, for example, in terms of generalization and robustness, and is an active area of research.

8. Challenges of Discovery

Arguably, one of the biggest challenges in discovering temporal abstractions arises from the **rich diversity of objectives** that the community has shown to yield meaningful options. As discussed throughout this paper, there are many distinct use cases for HRL, meaning methods are often aiming to achieve entirely different things. Because of this, it is frequently difficult to directly compare different methods across domains. This variety can be observed in the wide array of approaches presented in Sections 4, 5, and 6, begging the question whether there exists a single option discovery algorithm that could be generally applied.

Additionally, the **complexity overhead** that option discovery sometimes introduces can make it less appealing from a practical perspective (Jinnai et al., 2019b). The time invested by a practitioner in setting up an HRL algorithm is valuable. If this time investment does not lead to significantly improved performance on a particular task, or is not generally applicable across tasks, the practitioner will likely choose a simpler approach.

The two aforementioned points indicate that there is much potential for research in HRL to find **reliable and general** solutions as well as understanding where to apply them (see

Section 10). We now highlight prominent technical challenges that arise when attempting to discover temporal abstractions.

8.1 Non-Stationarity

One of the main difficulties in learning a hierarchy of behaviours stems from its modular nature. A hierarchical agent must learn, potentially simultaneously, about the option policies, option reward functions, termination functions, initiation functions, and high-level policy. These components interact and are typically defined in terms of each other, so the non-stationarity induced by learning one will create corresponding non-stationarity in the others.

A straightforward approach to deal with this non-stationarity is to learn the different components separately. For instance, this can be done by leveraging offline datasets (methods in Section 5) to first learn a set of option rewards or a set of option policies, before fixing them. These components can then be provided to a high-level policy that will learn to achieve a certain task. Similarly, we can leverage an LLM’s prior knowledge to define an option reward function or a policy directly (methods in Section 6). These would create stationary targets for the remainder of the components. LLMs can also be used to model the high-level policy itself, either by directly querying them or by leveraging their coding abilities to define the skill execution logic. The in-context learning abilities of LLMs could further allow for fast, gradient-free adaptation with respect to a changing option set.

When learning tabula rasa, non-stationarity between all the randomly initialized option components can be particularly challenging. It is common to first define an option learning phase, where the high-level policy acts according to a more exploratory behaviour, for example by uniformly choosing over the options (Machado et al., 2017; Eysenbach et al., 2019). Such a phase is meant to provide experience in learning the option reward functions and option policies. Nachum et al. (2018) emphasize the difficulty of non-stationarity in HRL when learning from past experiences that are stored in a dataset, called an experience replay buffer (Lin, 1991). An option that was previously sampled and stored within a replay buffer, together with the experience it generated, would not produce the same data distribution if we were to sample it now. To alleviate this, Nachum et al. (2018) relabel which option was used for a stored datapoint with the option that is currently most likely to generate the actions seen in this datapoint.

The non-stationarity challenge also affects the initiation function. Any method that learns initiation sets from option outcomes—for example, by classifying states according to whether option execution succeeded from them (Section 4.3)—implicitly evaluates the option policy as it stood when the data was collected; such binary classification (or, equivalently, Monte Carlo value estimation) is only sound when the option policy is fixed. When the policy is still improving, a vicious cycle emerges: when an option fails from a state, its initiation probability there decreases, making the agent less likely to execute the option from that state, which in turn deprives the option policy of the very data it needs to improve in that region. Left unchecked, initiation sets shrink over the agent’s lifetime, becoming overly conservative. Bagaria et al. (2023) diagnose this pathology and address it with tools from off-policy evaluation, adding exploration bonuses that increase the initiation probability of states from which policy improvement is most likely.

Non-stationarity, to a certain degree, has always been present in RL—for example, in actor-critic algorithms (Konda and Tsitsiklis, 2000). However, the large number of moving parts in a hierarchical agent makes it a fundamental challenge in HRL, and our understanding of it remains limited, both theoretically and empirically. Finding reliable option discovery methods will require both better ways to measure such instabilities and learning algorithms that remain stable in their presence.

8.2 Simultaneously Learning About Multiple Behaviours

If an agent has access to many options, it may solve a variety of tasks by re-composing option policies. However, such a large library comes at the cost of first learning the options themselves, highlighting some of the fundamental trade-offs presented in Section 2.2.

To approach this problem, it is convenient to turn to off-policy algorithms (Precup et al., 2000). Such algorithms allow for learning from data that was not generated by the current policy. Klissarov and Precup (2021) propose update rules to improve all options simultaneously by relying on a decomposition of the state-option distribution, introducing a minimal amount of off-policy corrections, and remaining compatible with any policy optimization algorithm. Their method can also be seen as an all-options policy optimization, similar to all-action updates in RL (Sutton et al., 2001). Daniel et al. (2016) instead leverage the perspective in which options are seen as latent codes. The authors adopt an expectation-maximization approach, which assumes a linear structure of the option policies. Smith et al. (2018) alleviate this assumption and derive a policy gradient objective that improves the data efficiency and interpretability of the learned options. Wulfmeier et al. (2020) leverages dynamic programming to infer option and action probabilities in hindsight.

We have previously introduced the methods of hindsight relabeling (Andrychowicz et al., 2017) as part of the skill discovery methods. We can reframe their approach through this question: if you have a multitude of options, or even a continuous spectrum, which other option should you update for a given trajectory? Hindsight experience replay answers this question by relabeling the trajectory stored in the replay buffer with the final state that was reached. This is an instance of off-policy learning as the experience generated by one policy is used to update another policy. The importance of learning off-policy through re-labeling in a hierarchy is further emphasized by Nachum et al. (2018) and Levy et al. (2019).

Is it possible to sample-efficiently learn about multiple options from a single stream of experience? Barreto et al. (2020) propose the Generalized Policy Improvement (GPI) update rules to answer this question (see also Section 7.2.2). The authors extend the concept of improvement from a single policy to multiple policies simultaneously. Specifically, their theorem states that, for a given set of policies, $\pi_1, \pi_2, \dots, \pi_n$, and their associated approximate Q values, $Q^{\pi_1}, Q^{\pi_2}, \dots, Q^{\pi_n}$,

$$\pi(s) \in \operatorname{argmax}_a \max_i Q^{\pi_i}(s, a), \quad (109)$$

then $Q^\pi(s, a) \geq \max_i Q^{\pi_i}(s, a)$. This update rule is used by Barreto et al. (2019) to efficiently learn how to execute a combination of options. Thakoor et al. (2022) further generalize the results beyond Markov policies, in particular, to options whose execution duration follows a geometric distribution. Recently, Chandrasekar and Machado (2026) have shown how options obtained from the graph Laplacian (see Section 4.2) can be used as the set of policies

(options) for GPI, also quantifying the benefits of learning state-dependent weights that combine such policies. The idea of learning efficiently about multiple policies is closely related to concepts such as the successor representations (Dayan, 1993) and successor features (Barreto et al., 2017), as well as other decompositions of the transition function (Touati et al., 2023).

8.3 Combining Rewards

When learning option policies through their option reward functions (Definition 3), we are faced with another important question: how should we balance between the option reward and the environmental reward? Dayan and Hinton (1993) argue that the lower-level policies should be agnostic to the environmental reward and learned only through the intrinsic one.¹⁸ Vezhnevets et al. (2017) take a softer approach and provide both rewards, possibly as the environmental reward contains rich information in the environments that were considered. In other cases, there is no intrinsic reward at all (Bacon et al., 2017). Sutton et al. (2023) investigate these questions from the perspective of planning and learning with options that do or do not respect the environmental reward. Reward-respecting options are much more effective when used for planning. Zahavy et al. (2022) propose a point of view of constrained optimization to balance these objectives and leverage Lagrange multipliers in practice. A thorough examination concerning the trade-offs of how hierarchical agents combine environmental reward and intrinsic reward has yet to be made.

9. Related Fields

We now discuss several fields adjacent to HRL. As noted earlier, HRL leverages temporal abstraction, and it is therefore important to understand how it relates to abstractions defined over other components of RL, such as state and action abstractions. Continual RL has also received increasing attention. Since continual-learning agents typically operate under limited capacity, abstraction can provide a principled mechanism for retaining and reusing knowledge, and temporal abstraction in particular offers one way to achieve this. Moreover, as discussed in Section 8.1, hierarchical agents can inherently introduce non-stationarity, suggesting that methods developed in continual RL may also benefit HRL. As we motivate HRL using the analogy of programming languages in Section 2, we also discuss the conceptually adjacent field of programmatic RL. Finally, HRL is closely connected to multi-agent RL. Many hierarchical systems can themselves be viewed as multi-agent systems, while hierarchical organization also naturally arises in multi-agent settings, particularly in cooperative multi-agent RL. These connections therefore warrant discussion.

9.1 State and Action Abstractions in Reinforcement Learning

Scaling RL for real-world applications faces challenges in handling high-dimensional or noisy observations and large action spaces. Accordingly, the RL community has long explored *abstractions*, a core computer science concept for suppressing low-level details so that reasoning can proceed at a higher conceptual level (Colburn and Shute, 2007). This can

18. Although this work predates the options framework (Sutton et al., 1999b), these lower-level policies can be viewed as option policies.

mitigate the curse of dimensionality and improve sample efficiency (Konidaris and Barto, 2009a; Ho et al., 2019; Abel, 2020; Ahmetoglu et al., 2026). The temporal abstractions introduced above provide one way to achieve this by reasoning over temporally extended behaviours rather than primitive actions. More generally, RL admits several other forms of abstraction that simplify different components of the learning problem. In this section, we review these forms of abstraction and the equivalence relations they induce.

State abstraction. It maps concrete states to compact representations while retaining information relevant to decision making. At the level of implementation, classical approaches construct explicit state aggregations (Singh et al., 1994; Li et al., 2006; Hutter, 2014), whereas deep RL approaches use neural networks to compress high-dimensional observations into learned representations (Gelada et al., 2019; Abel, 2020; Allen et al., 2021). Recent methods learn such representations via gradient-based optimization, introducing objectives such as reward prediction, self-prediction, inverse dynamics prediction, or metric losses to obtain compact and informative embeddings (Gelada et al., 2019; Allen et al., 2021; Ni et al., 2024; Luo et al., 2025).

Independently of how the representation is implemented, state-abstraction schemes differ in the behavioural information they preserve (Li et al., 2006). For example, some merge states only when they yield identical immediate rewards and transition dynamics under every action (which is called *model-preserving*), whereas others require equality of their optimal action-value functions. Bisimulation metrics relax these binary equivalence criteria by defining a graded behavioural distance based on discrepancies in rewards and transition dynamics (Ferns et al., 2004, 2011; Castro, 2020; Zhang et al., 2021a; Luo et al., 2025). Those metrics can support either state aggregation, by grouping states whose distance falls below a chosen threshold, or representation learning, by embedding states so that distances in the learned representation space approximate their behavioural distances (Luo et al., 2025).

State-action abstraction. Another line of work considers *state-action abstraction*, notably MDP homomorphisms, which map states and actions to an abstract MDP while preserving reward and transition dynamics (Ravindran and Barto, 2001; Ravindran, 2004; Ravindran and Barto, 2004; Narayanamurthy and Ravindran, 2008). Such abstractions can identify behaviourally equivalent state-action pairs, including those arising from symmetries in the underlying MDP. Similarly, this idea has been explored in deep RL literature (Rezaei-Shoshtari et al., 2022; Fujimoto et al., 2025).

Action abstraction. Throughout this paper, action abstraction is discussed primarily as temporal abstraction, whereby behaviours, usually spanning multiple timesteps, are represented as abstract actions. We devote less attention to per-timestep action abstraction, which reduces the computational burden of large action spaces through action elimination (Even-Dar et al., 2006; Zahavy et al., 2018), action embeddings or transformations (van Hasselt and Wiering, 2009; Dulac-Arnold et al., 2015; Jiang et al., 2023), and affordance-based filtering (Abel et al., 2014; Fulda et al., 2017; Khetarpal et al., 2020a). Affordances, for example, restrict the actions available in a state to those satisfying a specified intent or task-relevant criterion. A related but distinct form of abstraction operates over policies rather than the actions available at an individual decision point. *Policy abstraction* maps policies into a

compact space to facilitate policy selection or search and need not introduce temporal abstraction (Mutti et al., 2022; Tang et al., 2022; Zhang et al., 2023; Nabati et al., 2023), although it overlaps with HRL when the policies are temporally extended skills (Barreto et al., 2019).

Relationship to HRL. In addition to temporal abstraction, HRL can integrate of various types of state and action abstractions. In earlier work of HRL, two common forms of state abstraction are employed: first, state abstraction within the high-level controller, enabling learning or planning in a more tractable space. Feudal RL (Dayan and Hinton, 1993), as a prominent example, employs information hiding to abstract low-level details from the state observed by the manager. Second, state abstraction within the low-level controller, which speeds up option policy learning by constructing representations that are only relevant for that option (Konidaris and Barto, 2009a). State abstractions within MAXQ (Dietterich, 1998) and option models are natural examples, as options can be defined exclusively for states where the option is applicable.

Several HRL methods combine state and action abstractions with temporal abstraction to facilitate transfer across tasks. Relativized options use partial MDP homomorphisms to represent a family of structurally related options through a single reduced option MDP (Ravindran and Barto, 2002; Ravindran, 2003; Ravindran and Barto, 2003). When the option is invoked, the current state is mapped into this reduced model, and the selected abstract action is mapped back to an appropriate concrete action according to the relevant transformation. Portable options (Konidaris and Barto, 2007) pursue the same transfer objective but use a different mechanism: rather than selecting an explicit transformation at each invocation, they define the option policy, initiation set, and termination condition directly over agent-centered features whose meanings remain consistent across tasks. These options can therefore be reused across tasks. Castro and Precup (2010) pursue the same broad goal of transfer across MDPs, but use bisimulation distances to determine which actions from a source policy can be reused in a target task. Their analysis also extends to temporally extended actions, showing that options can provide a more effective unit of transfer than primitive actions.

Whereas these methods seek representations that allow skills to transfer across settings, another line of work asks how states should be abstracted once a set of skills is given. Konidaris et al. (2018) construct a symbolic state abstraction tailored to the available skills, mapping each low-level state to symbols that determine which skills are applicable and how they change the abstract state. More generally, Abel et al. (2020) characterize conditions under which a state abstraction and associated options preserve the ability to represent near-optimal policies. Each such option can initiate in any original state mapped to a particular abstract state and terminates upon reaching a state mapped to a different abstract state. Ahmetoglu et al. (2026) start from a given set of skills and construct an abstract state space that records which skills are available and predicts their rewards and transition outcomes. Each abstract state represents a probability distribution over states in the original MDP, and the resulting abstract MDP is Markov and approximately model-preserving.

9.2 Continual Reinforcement Learning

Continual reinforcement learning (CRL) is often introduced through environmental *non-stationarity*: components such as the transition function, reward function, state space, or action space may change over time (Khetarpal et al., 2022). While this formulation captures

many important settings, it places the emphasis on changes in the environment rather than on the learning process of the agent. An agent-centric view instead characterizes CRL as continual adaptation through interaction (Ring, 1994; Abel et al., 2023, 2024). Whereas conventional RL typically frames learning as the search for a solution to a specified problem, a continual learner may never reach a point at which learning should cease. Streams of tasks with unannounced boundaries are one instance of this broader setting, but are not its defining case.

The need for continual adaptation follows partly from the limited capacity of the agent. Rather than conditioning its decisions on the complete interaction history, a practical agent maintains an *agent state*: an incrementally updated internal summary that may include learned representations, value estimates, models, stored experiences, and reusable skills (Dong et al., 2022b; Kumar et al., 2025b). Because memory, computation, and representational capacity are bounded, the agent cannot preserve and process every potentially relevant aspect of its history. This perspective is sharpened by the *Big World Hypothesis*, according to which the world contains more useful information and behavioural opportunities than a bounded agent can represent (Javed and Sutton, 2024). Consequently, non-stationarity can be relative to the agent: even when the interaction process is stationary when conditioned on the complete history, it may appear non-stationary relative to the agent’s compressed internal state. The *stability-plasticity dilemma* should therefore not be reduced to a choice between remembering old experiences and prioritizing recent ones (Carpenter and Grossberg, 1987). Instead, stability concerns retaining information that remains useful for future behaviour, whereas plasticity concerns incorporating newly available information that is likely to be useful. Some forgetting is unavoidable, and may be beneficial; it becomes catastrophic when the discarded information continues to support effective behaviour.

HRL offers a principled agent-side mechanism for organizing this limited capacity. Temporally extended actions, such as options or skills, can represent recurring courses of action as reusable decision units and allow an agent to learn and act at multiple timescales (Sutton et al., 1999b). Such temporal abstractions may support continual adaptation by preserving useful behavioural structure, accelerating the acquisition of related behaviours, and enabling existing skills to be recombined or specialized for new situations (Machado et al., 2023; Klissarov and Machado, 2023). In restricted continual-learning settings, theoretical results also indicate that the absence of temporal abstraction can incur additional regret (Létourneau et al., 2023). To sum up, one of the fundamental reasons for the synergy between HRL and continual RL is that both fields rarely focus on optimally solving tasks. Instead, they are concerned about fast adaptation and transferability.

While promising, integrating HRL and continual RL presents open research challenges. As mentioned in Section 8, it is necessary to develop scalable skill discovery methods that can function in non-stationary settings, devise frameworks that jointly optimize for continual learning and HRL objectives, and design benchmarks and metrics for evaluating agents.

9.3 Programmatic Reinforcement Learning

As stated in Section 2, HRL could be viewed through an analogy with programming languages and formal systems. An example of this connection is *Hoare Logic* (Hoare, 1969), a formal system for assessing the correctness of imperative programs, which shares a similar structure

with the option framework (see Section 3.2) including initiation sets (pre-conditions), policies (commands), and termination conditions (post-conditions). Both frameworks facilitate reasoning about action sequences, thereby enhancing the structuring of complex decision-making processes. Research efforts have been undertaken to bridge the gap between HRL, programming languages, logic, and formal methods.

Programs as high-level policy. Early approaches, HAM (Parr and Russell, 1997) and PHAM (Andre and Russell, 2000) used hierarchies of partially specified finite-state machines (FSM) to structure policies. There are four types of states in HAMs: *Action* states execute actions, *Call* states execute subroutines, *Choice* states select subsequent states non-deterministically, and *Stop* states halt execution and return control to prior call states. This provides a prototype for early HRL methods, allowing for better compositionality, transferability (Andre and Russell, 2000), and state abstraction (Andre and Russell, 2002). More recent approaches use programs, specifically in domain-specific languages (DSLs), as high-level policies to guide lower-level RL agents. These are often called programmatic policies. Such an approach allows the system designer to inject biases that could, for example, improve sample efficiency over neural representations (Moraes et al., 2025).

Programs convey structured, interpretable, and unambiguous information, and their incorporation into the policy space can reduce the search space for the overall solution and offer a natural method for integrating prior knowledge symbolically. The structured representation of these programs allows one to decompose policies into options that can also be used to induce spaces that are more conducive to search (Moraes and Lelis, 2024; Moraes et al., 2025). In general, the programs can be either hand-crafted (Andreas et al., 2017; Sun et al., 2020), or automatically synthesized over a predefined syntactic (Carvalho et al., 2024) or semantic (latent) space (Yang et al., 2021b; Hasanbeig et al., 2021; Moraes et al., 2023; Moraes and Lelis, 2024). Such synthesis can be performed by parameterizing the program space, also known as *neuro-symbolic* (Sheth and Roy, 2023) approaches (Denil et al., 2017; Sohn et al., 2018; Trivedi et al., 2021; Zhao et al., 2021; Qiu and Zhu, 2022; Liu et al., 2023a; Lin et al., 2024), or by leveraging foundation models (Wang et al., 2023a; Klissarov et al., 2025a; Moraes et al., 2025). Learning search guidance for these spaces is also an active area of research (Medeiros et al., 2022; Aleixo and Lelis, 2023). The idea of decomposing policies into subprograms has also been explored even when the underlying policy is a neural network (Alikhasi and Lelis, 2024).

Programs to option rewards. Recent studies (Jothimurugan et al., 2019; Icarte et al., 2022; Furelos-Blanco et al., 2023; Venuto et al., 2024) demonstrate the feasibility of translating the formal languages (e.g., programs or FSMs) into pseudo-reward signals that facilitate RL training, conceptually connecting to the intrinsic rewards discussed in Section 4.8..

Distilling the neural policies to interpretable programs. A series of studies focuses on condensing an agent’s policy into more hierarchical, interpretable, and verifiable formats such as programs (Verma et al., 2018, 2019) or Decision Trees (Bastani et al., 2018), enhancing both lightness and clarity.

9.4 Cooperative Multi-Agent Reinforcement Learning

Cooperative multi-agent RL (Cooperative MARL) and HRL can be seen as conceptually connected: managing problem complexity using the structure of the problem. By breaking down large-scale tasks into more manageable subtasks, both approaches improve tractability and facilitate learning. In cooperative MARL, decomposition is achieved by distributing the decision-making process among multiple agents, whereas in HRL, it is accomplished through temporal abstraction. As an example, Feudal RL (Dayan and Hinton, 1993) can be viewed as a multi-agent system comprising managers and workers. This framework naturally extends to cooperative MARL settings (Ahilan and Dayan, 2019). Extensive research has explored the integration of HRL with cooperative MARL; Pateria et al. (2021) provide further details in Section 3.5 of their work.

10. Promising Domains for Hierarchical Reinforcement Learning

In this manuscript, we have examined a wide diversity of HRL approaches, each time highlighting the important ways in which they help decision-making through the benefits we laid out in Section 2.1. The vast body of research in HRL encompasses a wide spectrum of methods spanning multiple environments and domains. Under this diversity of approaches, a key question emerges: **in what domains should we expect HRL to be most effective?** An obvious criterion is that the domain comprises temporally extended tasks, such as those found in lifelong learning. As the task horizon decreases, the agent’s opportunities to exploit temporal abstractions become increasingly constrained. For example, decomposing short-horizon tasks into subtasks is likely to be less fruitful than long-horizon ones. However, can we go beyond this obvious criterion to predict the suitability of HRL methods?

We argue that task complexity alone is not a sufficient justification for employing HRL; rather, **HRL methods should be employed to exploit specific forms of structure**, namely modularity and compositionality. While Section 2.2 illustrates that HRL can efficiently yield good solutions under sample and computation budgets, making it attractive for complex environments where optimality is impractical, it should not be viewed merely as a fallback option when non-hierarchical RL fails. Positioning HRL simply as a last resort for “hard” tasks ignores the underlying mechanics of why these methods work. In reality, a complex environment lacking identifiable and repeated structure may see no benefit from HRL. Similarly, if the goal is only to solve a narrow task, the inherent overhead of learning a hierarchy would likely limit HRL’s advantages

HRL appears best suited for long-horizon environments that allow for a diversity of goals that share a structural overlap (whether these goals are defined by the environment or the agent itself). From this perspective, **open-ended systems are particularly promising domains for HRL methods**. Hughes et al. (2024) define an open-ended system as one that presents a constant flow of novel and possibly learnable goals. It is common in such systems that these goals share, to a degree, a common underlying structure, which makes HRL particularly appealing. Below, we showcase specific domains exemplifying these characteristics. This list is not exhaustive but rather serves to illustrate settings where HRL might excel.

Web Agents. The World Wide Web, a dynamic and ever-changing environment, presents a unique challenge for AI agents. The recent surge in interest has led to a variety of implementations of challenging domains, such as Android-in-the-Wild (Rawles et al., 2023) or WebArena (Zhou et al., 2024). Its near-infinite tasks and constantly evolving goals demand adaptability and the ability to decompose complex objectives into manageable subgoals. As mentioned in Section 3, even if the resulting agent is not hierarchical (i.e., does not explicitly carry a set of skills), learning to navigate the web through HRL methods, such as curriculum-based ones, can help address the sheer complexity of the web. Indeed, Web Agents must learn to navigate a constantly shifting landscape of information and services, adapting to new data, evolving user preferences, and the emergence of novel websites and services. Another important characteristic is that many tasks of interest share a lot of underlying structure, a key point of HRL. Overall, this complex and open-ended domain requires agents capable of learning, adapting, and generalizing across multiple timescales, ultimately revolutionizing how we interact with the online world.

Robotics. Robotics, with its emphasis on embodied intelligence and real-world interaction, presents a compelling domain for exploring the potential of HRL methods. The tasks robots face, from navigating complex environments to manipulating objects with dexterity, involve long horizons where, at each step, a low-level action is sampled from a continuous action space. HRL offers a natural framework for decomposing these complex tasks into manageable sub-policies, allowing robots to learn and refine abstract skills while also developing higher-level strategies for sequencing and coordinating them. Practical implementations of interest include AI2-THOR (Kolve et al., 2017), Habitat (Szot et al., 2021; Puig et al., 2024), CALVIN (Mees et al., 2022) and OGBench (Park et al., 2025).

The ability to recompose learned skills into novel combinations is crucial for robots operating in unstructured and dynamic environments, where adaptability and generalization are key. For instance, a robot learning to grasp objects might develop sub-policies for reaching, orienting its gripper, and applying the appropriate force. Ideally, these individual skills could then be recombined and adapted to grasp a wide variety of objects in different contexts, without requiring retraining from scratch. The long horizons inherent in many robotic tasks, coupled with the need for flexible and adaptable skill acquisition, make HRL a promising approach for developing robots capable of performing complex, real-world tasks with increasing autonomy and efficiency. Konidaris et al. (2011) used the methods discussed in Section 4.3 to provide the first demonstration of option discovery on a real robot; scaling such methods remains an important open problem.

Open-ended games. Training AI agents on games has a long history of striking successes in domains like Go (Silver et al., 2017) or Atari 2600 games (Mnih et al., 2015)¹⁹. However, for HRL to provide significant value over non-hierarchical RL, the domain would likely be complex, long-horizon, and open-ended. We have seen in Section 4.6 such an example, where a goal-conditioned policy trained on a large diversity of tasks led to human-timescale adaptation. Key to this success was the fact that data was readily available through fast simulation, allowing for quicker research iteration. This makes it particularly interesting to study open-ended games in order to better understand HRL methods. We now provide two

19. Note that this applies less to the case of hard exploration games such Montezuma’s Revenge, Private Eye, or Pitfall!, among others.

such examples. **NetHack** is a complex roguelike game, and it is a good environment for exploring the benefits of HRL. It has been brought to the RL community through the NetHack Learning Environment (Küttler et al., 2020). Its open-ended nature, procedurally generated dungeons, and long-horizon gameplay require exploration, planning, and adaptation across multiple timescales. Success requires both immediate tactical decisions and strategizing towards long-term goals, demanding credit assignment across extended temporal spans. The vast diversity of situations encountered also requires generalization, making HRL’s ability to learn reusable sub-policies and higher-level strategies particularly valuable. **Minecraft** (Johnson et al., 2016; Kanervisto et al., 2022), with its expansive, procedurally generated world and open-ended gameplay, presents a compelling testbed for HRL algorithms. The game requires navigating across diverse biomes, gathering resources, crafting tools, and structures, and ultimately, surviving and thriving. This requires planning and execution across multiple timescales. For instance, while the immediate goal might be chopping down a tree for wood, this action serves the higher-level objective of building a shelter for protection against nocturnal mobs. Furthermore, Minecraft’s crafting system inherently embodies a hierarchical structure. Creating complex items like diamond tools requires a chain of prerequisite crafting steps, each with its own subgoals and resource requirements. HRL agents could learn to decompose these complex tasks into manageable sub-policies, mirroring the hierarchical nature of crafting itself.

11. Conclusion

The ability to write, store, and reuse subroutines is one of the great ideas in computer science (Biermann, 1990). A subroutine, once written, can be called without regard to its internal operation; that is modularity. A subroutine can also be combined with others to generate new programs its author never anticipated; that is the power of compositionality. These two powerful ideas are the organizing principles that have enabled human engineers to write and maintain the extraordinarily complex programs that have powered the computing revolution. A generally intelligent AI agent will likely require similar organization, and HRL provides the foundations on which such agents can be built: procedural abstractions that improve exploration, credit assignment, and knowledge transfer. In this text, we looked at three broad families of approaches to acquiring such abstractions—discovery via online experience, via offline data, and via foundation models—organizing each around benefits and the central price it exacts.

No inductive bias is universally beneficial, and HRL’s bias toward decomposable structure can hurt as readily as help when the structure is misimagined. Hierarchies enter a Pareto frontier between performance, sample efficiency, and computational cost, often trading a little of the first and third for a great deal of the second.

The core open question before the field is that of discovery: how can we build agents that acquire these abstractions themselves via interaction? This problem of option discovery, in its grandest form, is one of the biggest meta-scientific questions of our time. An agent that can ask useful questions (option objectives) and efficiently find answers to them (option learning), will eventually acquire abstractions that unlock sample efficient reward maximization in many tasks.

This is an opportune moment to study this problem of option discovery. Advances in deep learning have made the problems on which hierarchy plausibly matters—high-dimensional and long-horizon—tractable to study at scale. Modern foundation models and large offline datasets offer means of incorporating useful priors into an agent’s learning, letting discovery begin from somewhere informed rather than from nothing. Together, these developments make the HRL problem feel within reach and well-poised to scale RL to larger, more complicated, and high-impact application areas.

Notation

Table 2: Glossary of notations used in RL and HRL (see Section 3).

Reinforcement Learning (RL)	
$S_t \in \mathcal{S}$	State at time step t
$A_t \in \mathcal{A}$	Action at time step t
R_{t+1}	Real-valued random variable representing the reward at time $t + 1$
$p(s' \mid s, a)$	Transition probability
ρ_0	Initial state distribution
$\gamma \in [0, 1)$	Discount factor
$\pi(a \mid s)$	Policy over actions
$d_\pi^\gamma(s)$	Discounted state occupancy under policy π
$q_\pi(s, a)$	Action-value function under policy π
$v_\pi(s)$	State-value function under policy π
$V_\phi(s)$	Estimated state-value function, parameterized by ϕ
$Q_\phi(s, a)$	Estimated action-value function, parameterized by ϕ
Hierarchical Reinforcement Learning (HRL)	
$o \in \mathcal{O}$	Option
$\pi_\theta(a \mid s, o)$	Option policy, parameterized by θ
$\beta_\psi(s, o)$	Option termination function, parameterized by ψ
$\mathcal{I}_\chi(s, o)$	Option initiation function, parameterized by χ
$\mu_\kappa(o \mid s)$	High-level policy, parameterized by κ
$P_\mathcal{O}(s, o, s')$	Option transition model
$R_\mathcal{O}(s, o)$	Option reward model
$q_\pi(s, o)$	Option-value function under policy π
$q_u(s, o, a)$	Action-value within the context of option o
$u_\beta(o, s')$	Option value upon arrival
$v_\mu(s)$	Value function under high-level policy μ
$\omega_o \in \Omega$	Option objective associated with option o
$r_\nu^o(s, a)$	Option reward function for option o , parameterized by ν
$z \in \mathcal{Z}$	Option identifier
$g \in \mathcal{G}$	Goal or subgoal

Acknowledgements

We would like to thank Xujie Si, Khimya Khetarpal, Seohong Park, Bartłomiej Cupiał, Isabeau Prémont-Schwarz, Levi Lelis, Andrew Levy, Alex Ivanov, Nishanth Anand, and Jonathan Colaço Carr for their valuable feedback. This research is supported in part by NSF 1955361 and NSF CAREER 1844960. The research is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and the Canada CIFAR AI Chair Program.

References

- Abel, D. (2020). *A Theory of Abstraction in Reinforcement Learning*. PhD thesis, Brown University.
- Abel, D., Barreto, A., Van Roy, B., Precup, D., van Hasselt, H. P., and Singh, S. P. (2023). A definition of continual reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Abel, D., Barth-Maron, G., MacGlashan, J., and Tellex, S. (2014). Toward affordance-aware planning. In *First Workshop on Affordances: Affordances in Vision for Cognitive Robotics*.
- Abel, D., Ho, M. K., and Harutyunyan, A. (2024). Three dogmas of reinforcement learning. *Reinforcement Learning Journal*, 2:629–644.
- Abel, D., Umbanhowar, N., Khetarpal, K., Arumugam, D., Precup, D., and Littman, M. L. (2020). Value preserving state-action abstractions. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.
- Abramson, D. I. A. T. J., Ahuja, A., Brussee, A., Carnevale, F., Cassin, M., Fischer, F., Georgiev, P., Goldin, A., Harley, T., Hill, F., Humphreys, P. C., Hung, A., Landon, J., Lillicrap, T. P., Merzic, H., Muldal, A., Santoro, A., Scully, G., von Glehn, T., Wayne, G., Wong, N., Yan, C., and Zhu, R. (2021). Creating multimodal interactive agents with imitation and self-supervised learning. *CoRR*, abs/2112.03763.
- Achiam, J., Edwards, H., Amodei, D., and Abbeel, P. (2018). Variational option discovery algorithms. *CoRR*, abs/1807.10299.
- Adeniji, A., Xie, A., Sferrazza, C., Seo, Y., James, S., and Abbeel, P. (2023). Language reward modulation for pretraining reinforcement learning. *CoRR*, abs/2308.12270.
- Ahlan, S. and Dayan, P. (2019). Feudal multi-agent hierarchies for cooperative reinforcement learning. *CoRR*, abs/1901.08492.
- Ahmetoglu, A., James, S., Allen, C., Lobel, S., Abel, D., and Konidaris, G. (2026). Skill-driven neurosymbolic state abstractions. In *Neural Information Processing Systems (NeurIPS)*.
- Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Ho, D., Hsu, J., Ibarz, J., Ichter, B., Irpan,

- A., Jang, E., Ruano, R. J., Jeffrey, K., Jesmonth, S., Joshi, N., Julian, R., Kalashnikov, D., Kuang, Y., Lee, K.-H., Levine, S., Lu, Y., Luu, L., Parada, C., Pastor, P., Quiambao, J., Rao, K., Rettinghouse, J., Reyes, D., Sermanet, P., Sievers, N., Tan, C., Toshev, A. T., Vanhoucke, V., Xia, F., Xiao, T., Xu, P., Xu, S., Yan, M., and Zeng, A. (2022). Do as I can and not as I say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*.
- Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows - Theory, Algorithms and Applications*. Prentice Hall.
- Ajay, A., Kumar, A., Agrawal, P., Levine, S., and Nachum, O. (2021). OPAL: Offline primitive discovery for accelerating offline reinforcement learning. In *International Conference on Learning Representations (ICLR)*.
- Akakzia, A., Colas, C., Oudeyer, P.-Y., Chetouani, M., and Sigaud, O. (2021). Grounding language to autonomously-acquired skills via goal generation. In *International Conference on Learning Representations (ICLR)*.
- Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., Ring, R., Rutherford, E., Cabi, S., Han, T., Gong, Z., Samangooei, S., Monteiro, M., Menick, J., Borgeaud, S., Brock, A., Nematzadeh, A., Sharifzadeh, S., Binkowski, M., Barreira, R., Vinyals, O., Zisserman, A., and Simonyan, K. (2022). Flamingo: A visual language model for few-shot learning. In *Neural Information Processing Systems (NeurIPS)*.
- Aleixo, D. S. and Lelis, L. H. S. (2023). Show me the way! Bilevel search for synthesizing programmatic strategies. In *AAAI Conference on Artificial Intelligence*.
- Alikhasi, M. and Lelis, L. H. S. (2024). Unveiling options with neural network decomposition. In *International Conference on Learning Representations (ICLR)*.
- Allen, C., Parikh, N., Gottesman, O., and Konidaris, G. (2021). Learning Markov state abstractions for deep reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Ames, B. and Konidaris, G. (2019). Bounded-error LQR-Trees. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Amin, S., Gomrokchi, M., Satija, H., van Hoof, H., and Precup, D. (2021). A survey of exploration methods in reinforcement learning. *CoRR*, abs/2109.00157.
- Amodei, D., Olah, C., Steinhardt, J., Christiano, P. F., Schulman, J., and Mané, D. (2016). Concrete problems in AI safety. *CoRR*, abs/1606.06565.
- Amortila, P., Foster, D. J., and Krishnamurthy, A. (2024). Scalable online exploration via coverability. In *International Conference on Machine Learning (ICML)*.
- Andre, D. and Russell, S. J. (2000). Programmable reinforcement learning agents. In *Neural Information Processing Systems (NeurIPS)*.

- Andre, D. and Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. In *AAAI Conference on Artificial Intelligence*.
- Andreas, J., Klein, D., and Levine, S. (2017). Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning (ICML)*.
- Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. In *Neural Information Processing Systems (NeurIPS)*.
- Anthropic (2024). Developing a computer use model. <https://www.anthropic.com/news/developing-computer-use>.
- Arjona-Medina, J. A., Gillhofer, M., Widrich, M., Unterthiner, T., Brandstetter, J., and Hochreiter, S. (2019). RUDDER: Return decomposition for delayed rewards. In *Neural Information Processing Systems (NeurIPS)*.
- Auer, P., Jaksch, T., and Ortner, R. (2008). Near-optimal regret bounds for reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Bacon, P. and Precup, D. (2016). A matrix splitting perspective on planning with options. *CoRR*, abs/1612.00916.
- Bacon, P.-L. (2013). On the bottleneck concept for options discovery: Theoretical underpinnings and extension in continuous state spaces. Master’s thesis, McGill University.
- Bacon, P.-L. (2018). *Temporal Representation Learning*. PhD thesis, McGill University.
- Bacon, P.-L., Harb, J., and Precup, D. (2017). The option-critic architecture. *The Association for the Advancement of Artificial Intelligence*.
- Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. (2020a). Agent57: Outperforming the Atari human benchmark. In *International Conference on Machine Learning (ICML)*.
- Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., and Blundell, C. (2020b). Never give up: Learning directed exploration strategies. In *International Conference on Learning Representations (ICLR)*.
- Bagaria, A. (2025). *Skill Discovery for Exploration and Planning*. PhD thesis, Brown University.
- Bagaria, A., Abbatematteo, B. M., Gottesman, O., Corsaro, M., Rammohan, S., and Konidaris, G. (2023). Effectively learning initiation sets in hierarchical reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Bagaria, A., Koch, A. D. M., and Konidaris, G. (2026). Going beyond state-reaching: Learning abstractions for intrinsically motivated skill discovery. In *Reinforcement Learning in Big Worlds Workshop, Reinforcement Learning Conference (RLC)*.

- Bagaria, A., Koch, A. D. M., Rodriguez-Sanchez, R., Lobel, S., and Konidaris, G. (2025). Intrinsically motivated discovery of temporally abstract graph-based models of the world. *Reinforcement Learning Journal*.
- Bagaria, A. and Konidaris, G. (2020). Option discovery using deep skill chaining. In *International Conference on Learning Representations (ICLR)*.
- Bagaria, A. and Schaul, T. (2023). Scaling goal-based exploration via pruning proto-goals. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Bagaria, A., Senthil, J. K., and Konidaris, G. (2021a). Skill discovery for exploration and planning using deep skill graphs. In *International Conference on Machine Learning (ICML)*.
- Bagaria, A., Senthil, J. K., Slivinski, M., and Konidaris, G. (2021b). Robustly learning composable options in deep reinforcement learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Bahdanau, D., Hill, F., Leike, J., Hughes, E., Hosseini, S., Kohli, P., and Grefenstette, E. (2018). Learning to understand goal specifications by modelling reward. In *International Conference on Learning Representations (ICLR)*.
- Bai, H., Zhou, Y., Cemri, M., Pan, J., Suhr, A., Levine, S., and Kumar, A. (2024). DigiRL: Training in-the-wild device-control agents with autonomous reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Bai, Y., Kadavath, S., Kundu, S., Askill, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., Kerr, J., Mueller, J., Ladish, J., Landau, J., Ndousse, K., Lukošiu̇tė, K., Lovitt, L., Sellitto, M., Elhage, N., Schiefer, N., Mercado, N., DasSarma, N., Lasenby, R., Larson, R., Ringer, S., Johnston, S., Kravec, S., Showk, S. E., Fort, S., Lanham, T., Telleen-Lawton, T., Conerly, T., Henighan, T. J., Hume, T., Bowman, S., Hatfield-Dodds, Z., Mann, B., Amodei, D., Joseph, N., McCandlish, S., Brown, T. B., and Kaplan, J. (2022). Constitutional AI: Harmlessness from AI feedback. *CoRR*, abs/2212.08073.
- Baranes, A. and Oudeyer, P.-Y. (2013). Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61:49–73.
- Barreto, A., Borsa, D., Hou, S., Comanici, G., Aygün, E., Hamel, P., Toyama, D., Hunt, J. J., Mourad, S., Silver, D., and Precup, D. (2019). The option keyboard: Combining skills in reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Barreto, A., Borsa, D., Quan, J., Schaul, T., Silver, D., Hessel, M., Mankowitz, D. J., Zidek, A., and Munos, R. (2018). Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *International Conference on Machine Learning (ICML)*.

- Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. (2017). Successor features for transfer in reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Barreto, A., Hou, S., Borsa, D., Silver, D., and Precup, D. (2020). Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117:30079–30087.
- Barto, A. G. and Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13:41–77.
- Barto, A. G. and Şimşek, Ö. (2005). Intrinsic motivation for reinforcement learning systems. In *thirteenth yale workshop on adaptive and learning systems*.
- Barto, A. G., Singh, S. P., and Chentanez, N. (2004). Intrinsically motivated learning of hierarchical collections of skills. In *International Conference on Development and Learning (ICDL)*.
- Bastani, O., Pu, Y., and Solar-Lezama, A. (2018). Verifiable reinforcement learning via policy extraction. In *Neural Information Processing Systems (NeurIPS)*.
- Bauer, J., Baumli, K., Behbahani, F. M. P., Bhoopchand, A., Bradley-Schmieg, N., Chang, M., Clay, N., Collister, A., Dasagi, V., Gonzalez, L., Gregor, K., Hughes, E., Kashem, S., Loks-Thompson, M., Openshaw, H., Parker-Holder, J., Pathak, S., Nieves, N. P., Rakicevic, N., Rocktäschel, T., Schroecker, Y., Singh, S. P., Sygnowski, J., Tuyls, K., York, S., Zacherl, A., and Zhang, L. M. (2023). Human-timescale adaptation in an open-ended task space. In *International Conference on Machine Learning (ICML)*.
- Baumli, K., Baveja, S., Behbahani, F. M. P., Chan, H., Comanici, G., Flennerhag, S., Gazeau, M., Holsheimer, K., Horgan, D., Laskin, M., Lyle, C., Masoom, H., McKinney, K., Mnih, V., Neitz, A., Pardo, F., Parker-Holder, J., Quan, J., Rocktaschel, T., Sahni, H., Schaul, T., Schroecker, Y., Spencer, S., Steigerwald, R., Wang, L., and Zhang, L. (2023). Vision-language models as a source of rewards. *CoRR*, abs/2312.09187.
- Baumli, K., Warde-Farley, D., Hansen, S., and Mnih, V. (2021). Relative variational intrinsic control. In *AAAI Conference on Artificial Intelligence*.
- Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., Legg, S., and Petersen, S. (2016). DeepMind lab. *CoRR*, abs/1612.03801.
- Beck, J., Vuorio, R., Zheran Liu, E., Xiong, Z., Zintgraf, L., Finn, C., and Whiteson, S. (2025). A tutorial on meta-reinforcement learning. *Foundations and Trends in Machine Learning*, 18(2-3):224–384.
- Bellemare, M. G., Candido, S., Castro, P. S., Gong, J., Machado, M. C., Moitra, S., Ponda, S. S., and Wang, Z. (2020). Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588:77–82.

- Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. H. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.
- Bellemare, M. G., Ostrovski, G., Guez, A., Thomas, P. S., and Munos, R. (2016a). Increasing the action gap: New operators for reinforcement learning. In *AAAI Conference on Artificial Intelligence*.
- Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016b). Unifying count-based exploration and intrinsic motivation. In *Neural Information Processing Systems (NeurIPS)*.
- Bellman, R. (1957). Dynamic programming. *Science*, 153:34–37.
- Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *International Conference on Machine Learning (ICML)*.
- Berlyne, D. E. (1960). *Conflict, Arousal and Curiosity*. McGraw Hill.
- Bertsekas, D. P. (1995). *Dynamic Programming and Optimal Control, Two Volume Set*. Athena Scientific.
- Biermann, A. W. (1990). *Great Ideas in Computer Science: A Gentle Introduction*. MIT Press.
- Boutilier, C., Dearden, R., and Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107.
- Bradley, R. A. and Terry, M. E. (1952). Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39:324.
- Bradtke, S. J. and Duff, M. O. (1994). Reinforcement learning methods for continuous-time Markov decision problems. In *Neural Information Processing Systems (NeurIPS)*.
- Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Chen, X., Choromanski, K., Ding, T., Driess, D., Dubey, A., Finn, C., Florence, P., Fu, C., Arenas, M. G., Gopalakrishnan, K., Han, K., Hausman, K., Herzog, A., Hsu, J., Ichter, B., Irpan, A., Joshi, N., Julian, R., Kalashnikov, D., Kuang, Y., Leal, I., Lee, L., Lee, T.-W. E., Levine, S., Lu, Y., Michalewski, H., Mordatch, I., Pertsch, K., Rao, K., Reymann, K., Ryoo, M., Salazar, G., Sanketi, P., Sermanet, P., Singh, J., Singh, A., Soricut, R., Tran, H., Vanhoucke, V., Vuong, Q., Wahid, A., Welker, S., Wohlhart, P., Wu, J., Xia, F., Xiao, T., Xu, P., Xu, S., Yu, T., and Zitkovich, B. (2023). RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning (CoRL)*.
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T. J., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In *Neural Information Processing Systems (NeurIPS)*.

- Brunskill, E. and Li, L. (2014). PAC-inspired option discovery in lifelong reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2019). Exploration by random network distillation. In *International Conference on Learning Representations (ICLR)*.
- Burridge, R. R., Rizzi, A. A., and Koditschek, D. E. (1999). Sequential composition of dynamically dexterous robot behaviors. *International Journal Robotics Research*, 18(6):534–555.
- Campero, A., Raileanu, R., Kuttler, H., Tenenbaum, J. B., Rocktäschel, T., and Grefenstette, E. (2021). Learning with AMIGo: Adversarially motivated intrinsic goals. In *International Conference on Learning Representations (ICLR)*.
- Campos, V., Trott, A., Xiong, C., Socher, R., Giro-I-Nieto, X., and Torres, J. (2020). Explore, discover and learn: Unsupervised discovery of state-covering skills. In *International Conference on Machine Learning (ICML)*.
- Carpenter, G. A. and Grossberg, S. (1987). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37(1):54–115.
- Carvalho, T. H., Tjhia, K., and Lelis, L. H. S. (2024). Reclaiming the source of programmatic policies: Programmatic versus latent spaces. In *International Conference on Learning Representations (ICLR)*.
- Carvalho, W. T., Filos, A., Lewis, R., Lee, H., and Singh, S. P. (2023). Composing task knowledge with modular successor feature approximators. In *International Conference on Learning Representations (ICLR)*.
- Castro, P. S. (2020). Scalable methods for computing state similarity in deterministic Markov decision processes. In *AAAI Conference on Artificial Intelligence*.
- Castro, P. S. and Precup, D. (2010). Using bisimulation for policy transfer in MDPs. In *AAAI Conference on Artificial Intelligence*.
- Chandrasekar, S. and Machado, M. C. (2026). The Laplacian keyboard: Beyond the linear span. *CoRR*, abs/2602.07730.
- Chen, B., Zhu, C., Agrawal, P., Zhang, K., and Gupta, A. (2023). Self-supervised reinforcement learning that transfers using random features. In *Neural Information Processing Systems (NeurIPS)*.
- Chevalier-Boisvert, M., Dai, B., Towers, M., Perez-Vicente, R. D. L., Willems, L., Lahlou, S., Pal, S., Castro, P. S., and Terry, J. K. (2023). Minigrid & Miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. In *Neural Information Processing Systems (NeurIPS)*.
- Choi, J., Sharma, A., Lee, H., Levine, S., and Gu, S. S. (2021). Variational empowerment as representation learning for goal-based reinforcement learning. In *International Conference on Machine Learning (ICML)*.

- Chua, R., Ghosh, A., Kaplanis, C., Richards, B. A., and Precup, D. (2024). Learning successor features the simple way. In *Neural Information Processing Systems (NeurIPS)*.
- Chung, F. R. (1997). *Spectral Graph Theory*. American Mathematical Society.
- Colas, C., Akakzia, A., Oudeyer, P.-Y., Chetouani, M., and Sigaud, O. (2020a). Language-conditioned goal generation: A new approach to language grounding for RL. *CoRR*, abs/2006.07043.
- Colas, C., Karch, T., Lair, N., Dussoux, J.-M., Moulin-Frier, C., Dominey, P. F., and Oudeyer, P.-Y. (2020b). Language as a cognitive tool to imagine goals in curiosity-driven exploration. In *Neural Information Processing Systems (NeurIPS)*.
- Colas, C., Karch, T., Sigaud, O., and Oudeyer, P.-Y. (2022). Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: A short survey. *Journal of Artificial Intelligence Research*, 74:1159–1199.
- Colas, C., Oudeyer, P.-Y., Sigaud, O., Fournier, P., and Chetouani, M. (2018). CURIOUS: Intrinsically motivated modular multi-goal reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Colas, C., Teodorescu, L., Oudeyer, P.-Y., Yuan, X., and Côté, M.-A. (2023). Augmenting autotelic agents with large language models. In *Conference on Lifelong Learning Agents (CoLLAs)*.
- Colburn, T. R. and Shute, G. M. (2007). Abstraction in computer science. *Minds and Machines*, 17(2):169–184.
- Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games*.
- Cui, Y., Niekum, S., Gupta, A., Kumar, V., and Rajeswaran, A. (2022). Can foundation models perform zero-shot task specification for robot manipulation? In *Conference on Learning for Dynamics and Control (L4DC)*.
- da Silva, B. C., Konidaris, G., and Barto, A. G. (2012). Learning parameterized skills. In *International Conference on Machine Learning (ICML)*.
- Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. In *Association for Computational Linguistics (ACL)*.
- Daniel, C., van Hoof, H., Peters, J., and Neumann, G. (2016). Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104:337–357.
- Dayan, P. (1993). Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624.
- Dayan, P. and Hinton, G. E. (1993). Feudal reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.

- de Cothi, W. and Barry, C. (2020). Neurobiological successor features for spatial navigation. *Hippocampus*, 30(12):1347–1355.
- Denil, M., Colmenarejo, S. G., Cabi, S., Saxton, D., and De Freitas, N. (2017). Programmable agents. In *Neural Information Processing Systems (NeurIPS)*.
- Dennis, M., Jaques, N., Vinitzky, E., Bayen, A. M., Russell, S. J., Critch, A., and Levine, S. (2020). Emergent complexity and zero-shot transfer via unsupervised environment design. In *Neural Information Processing Systems (NeurIPS)*.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.
- Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Dong, Q., Li, L., Dai, D., Zheng, C., Wu, Z., Chang, B., Sun, X., Xu, J., and Sui, Z. (2022a). A survey on in-context learning. *Empirical Methods in Natural Language Processing*.
- Dong, S., Van Roy, B., and Zhou, Z. (2022b). Simple agent, complex environment: Efficient reinforcement learning with agent states. *Journal of Machine Learning Research*, 23(255):1–54.
- Du, Y., Konyushkova, K., Denil, M., Raju, A. S., Landon, J., Hill, F., de Freitas, N., and Cabi, S. (2023a). Vision-language models as success detectors. In *Conference on Lifelong Learning Agents (CoLLAs)*.
- Du, Y., Kosoy, E., Dayan, A., Rufova, M., Abbeel, P., and Gopnik, A. (2023b). What can AI learn from human exploration? Intrinsically-motivated humans and agents in open-world exploration. In *Intrinsically-Motivated and Open-Ended Learning Workshop, Neural Information Processing Systems (NeurIPS)*.
- Du, Y., Watkins, O., Wang, Z., Colas, C., Darrell, T., Abbeel, P., Gupta, A., and Andreas, J. (2023c). Guiding pretraining in reinforcement learning with large language models. In *International Conference on Machine Learning (ICML)*.
- Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). RL²: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779.
- Dulac-Arnold, G., Evans, R., van Hasselt, H. P., Sunehag, P., Lillicrap, T. P., Hunt, J. J., Mann, T., Weber, T., Degris, T., and Coppin, B. (2015). Deep reinforcement learning in large discrete action spaces. *CoRR*, abs/1512.07679.
- Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2020). First return, then explore. *Nature*, 590:580–586.
- Erraqabi, A., Machado, M. C., Zhao, M., Sukhbaatar, S., Lazaric, A., Denoyer, L., and Bengio, Y. (2022). Temporal abstractions-augmented temporally contrastive learning: An alternative to the Laplacian in RL. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

- Evans, J. B. and Şimşek, Ö. (2023). Creating multi-level skill hierarchies in reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Even-Dar, E., Mannor, S., Mansour, Y., and Mahadevan, S. (2006). Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Journal of machine learning research*, 7(6).
- Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. (2019). Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations (ICLR)*.
- Faldor, M., Zhang, J., Cully, A., and Clune, J. (2025). OMNI-EPIC: Open-endedness via models of human notions of interestingness with environments programmed in code. In *International Conference on Learning Representations (ICLR)*.
- Fan, L., Wang, G., Jiang, Y., Mandlekar, A., Yang, Y., Zhu, H., Tang, A., Huang, D., Zhu, Y., and Anandkumar, A. (2022). MineDojo: Building open-ended embodied agents with internet-scale knowledge. In *Neural Information Processing Systems (NeurIPS)*.
- Fang, M., Zhou, C., Shi, B., Gong, B., Xu, J., and Zhang, T. (2018). DHER: Hindsight experience replay for dynamic goals. In *International Conference on Learning Representations (ICLR)*.
- Fang, M., Zhou, T., Du, Y., Han, L., and Zhang, Z. (2019). Curriculum-guided hindsight experience replay. In *Neural Information Processing Systems (NeurIPS)*.
- Farahmand, A.-m. (2011). Action-gap phenomenon in reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Farebrother, J., Greaves, J., Agarwal, R., Lan, C. L., Goroshin, R., Castro, P. S., and Bellemare, M. G. (2023). Proto-value networks: Scaling representation learning with auxiliary tasks. In *International Conference on Learning Representations (ICLR)*.
- Farebrother, J., Pirotta, M., Tirinzoni, A., Bellemare, M. G., Lazaric, A., and Touati, A. (2026). Compositional planning with jumpy world models. In *ICLR 2026 the 2nd Workshop on World Models: Understanding, Modelling and Scaling*.
- Farebrother, J., Pirotta, M., Tirinzoni, A., Munos, R., Lazaric, A., and Touati, A. (2025). Temporal difference flows. In *International Conference on Machine Learning (ICML)*.
- Ferns, N., Panangaden, P., and Precup, D. (2004). Metrics for finite Markov decision processes. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.
- Ferns, N., Panangaden, P., and Precup, D. (2011). Bisimulation metrics for continuous Markov decision processes. *SIAM Journal on Computing*, 40(6):1662–1714.
- Fikes, R. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208.
- Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*.

- Florensa, C., Held, D., Geng, X., and Abbeel, P. (2018). Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning (ICML)*.
- Ford, L. R. and Fulkerson, D. R. (1962). *Flows in Networks*. Princeton University Press.
- Forestier, S., Mollard, Y., and Oudeyer, P.-Y. (2017). Intrinsically motivated goal exploration processes with automatic curriculum learning. *J. Mach. Learn. Res.*, 23:152:1–152:41.
- Forestier, S. and Oudeyer, P.-Y. (2016). Modular active curiosity-driven discovery of tool use. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Fox, R., Krishnan, S., Stoica, I., and Goldberg, K. (2017). Multi-level discovery of deep options. *CoRR*, abs/1703.08294.
- Frans, K., Ho, J., Chen, X., Abbeel, P., and Schulman, J. (2018). Meta learning shared hierarchies. In *International Conference on Learning Representations (ICLR)*.
- Fruit, R. and Lazaric, A. (2017). Exploration-exploitation in MDPs with options. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.
- Fu, H., Yu, S., Tiwari, S., Littman, M. L., and Konidaris, G. (2023). Meta-learning parameterized skills. In *International Conference on Machine Learning (ICML)*.
- Fu, J., Korattikara, A., Levine, S., and Guadarrama, S. (2019). From language to goals: Inverse reinforcement learning for vision-based instruction following. In *International Conference on Learning Representations (ICLR)*.
- Fu, Y., Zhang, H., Wu, D., Xu, W., and Boulet, B. (2024). FuRL: Visual-language models as fuzzy rewards for reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Fujimoto, S., D’Oro, P., Zhang, A., Tian, Y., and Rabbat, M. (2025). Towards general-purpose model-free reinforcement learning. In *International Conference on Learning Representations (ICLR)*.
- Fulda, N., Ricks, D., Murdoch, B., and Wingate, D. (2017). What can you do with a rock? Affordance extraction via word embeddings. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Furelos-Blanco, D., Law, M., Jonsson, A., Broda, K., and Russo, A. (2023). Hierarchies of reward machines. In *International Conference on Machine Learning (ICML)*.
- Gao, C.-X., Wu, C., Cao, M., Kong, R., Zhang, Z., and Yu, Y. (2024). ACT: Empowering decision transformer with dynamic programming via advantage conditioning. In *AAAI Conference on Artificial Intelligence*.
- Gelada, C., Kumar, S., Buckman, J., Nachum, O., and Bellemare, M. G. (2019). DeepMDP: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning (ICML)*.

- Gomez, D., Bowling, M. H., Lee, J. Q., and Machado, M. C. (2025). Representations in the hippocampal-entorhinal system emerge from learning sensory predictions. *bioRxiv*, 2025.10.03.680189.
- Gomez, D., Bowling, M. H., and Machado, M. C. (2023). Proper Laplacian representation learning. In *International Conference on Learning Representations (ICLR)*.
- Gopnik, A. (2024). Empowerment as causal learning, causal learning as empowerment: A bridge between bayesian causal hypothesis testing and reinforcement learning. In *Intrinsically Motivated Open-ended Learning Workshop, Neural Information Processing Systems (NeurIPS)*.
- Gopnik, A., Meltzoff, A., and Kuhl, P. (2009). *The Scientist in the Crib: What Early Learning Tells Us About the Mind*. HarperCollins.
- Gopnik, A. and Wellman, H. M. (2012). Reconstructing constructivism: Causal models, bayesian learning mechanisms, and the theory theory. *Psychological bulletin*, 138(6):1085.
- Grauman, K., Westbury, A., Byrne, E., Chavis, Z., Furnari, A., Girdhar, R., Hamburger, J., Jiang, H., Liu, M., Liu, X., Martin, M., Nagarajan, T., Radosavovic, I., Ramakrishnan, S. K., Ryan, F., Sharma, J., Wray, M., Xu, M., Xu, E. Z., Zhao, C., Bansal, S., Batra, D., Cartillier, V., Crane, S., Do, T., Doulaty, M., Erapalli, A., Feichtenhofer, C., Fragomeni, A., Fu, Q., Fuegen, C., Gebreselasie, A., Gonzalez, C., Hillis, J., Huang, X., Huang, Y., Jia, W., Khoo, W., Kolar, J., Kottur, S., Kumar, A., Landini, F., Li, C., Li, Y., Li, Z., Mangalam, K., Modhugu, R., Munro, J., Murrell, T., Nishiyasu, T., Price, W., Puentes, P. R., Ramazanov, M., Sari, L., Somasundaram, K., Southerland, A., Sugano, Y., Tao, R., Vo, M., Wang, Y., Wu, X., Yagi, T., Zhu, Y., Arbelaez, P., Crandall, D., Damen, D., Farinella, G. M., Ghanem, B., Ithapu, V. K., Jawahar, C. V., Joo, H., Kitani, K., Li, H., Newcombe, R., Oliva, A., Park, H. S., Rehg, J. M., Sato, Y., Shi, J., Shou, M. Z., Torralba, A., Torresani, L., Yan, M., and Malik, J. (2022). Ego4D: Around the world in 3,000 hours of egocentric video. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Gregor, K., Rezende, D. J., and Wierstra, D. (2017). Variational intrinsic control. In *International Conference on Learning Representations (ICLR)*.
- Guo, Z. D., Thakoor, S., Pislari, M., Pires, B. Á., Alth’e, F., Tallec, C., Saade, A., Calandriello, D., Grill, J.-B., Tang, Y., Valko, M., Munos, R., Azar, M. G., and Piot, B. (2022). Byol-explore: Exploration by bootstrapped prediction. *CoRR*, abs/2206.08332.
- Gupta, A., Kumar, V., Lynch, C., Levine, S., and Hausman, K. (2019). Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Conference on Robot Learning (CoRL)*.
- Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., and Levine, S. (2018). Meta-reinforcement learning of structured exploration strategies. In *Neural Information Processing Systems (NeurIPS)*.

- Haarnoja, T., Pong, V. H., Zhou, A., Dalal, M., Abbeel, P., and Levine, S. (2018). Composable deep reinforcement learning for robotic manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*.
- Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning (ICML)*.
- Hafner, D., Lee, K.-H., Fischer, I., and Abbeel, P. (2022). Deep hierarchical planning from pixels. In *Neural Information Processing Systems (NeurIPS)*.
- Hafner, D., Ortega, P. A., Ba, J., Parr, T., Friston, K. J., and Heess, N. (2020). Action and perception as divergence minimization. In *International Conference on Learning Representations (ICLR)*.
- Hansen, S., Dabney, W., Barreto, A., Warde-Farley, D., de Wiele, T. V., and Mnih, V. (2020). Fast task inference with variational intrinsic successor features. In *International Conference on Learning Representations (ICLR)*.
- Harb, J., Bacon, P.-L., Klissarov, M., and Precup, D. (2018). When waiting is not an option : Learning options with a deliberation cost. *AAAI Conference on Artificial Intelligence*.
- Harlow, H. F. (1950). Learning and satiation of response in intrinsically motivated complex puzzle performance by monkeys. *Journal of Comparative and Physiological Psychology*, 43(4):289–294.
- Harutyunyan, A., Dabney, W., Borsa, D., Heess, N., Munos, R., and Precup, D. (2019a). The termination critic. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.
- Harutyunyan, A., Vrancx, P., Hamel, P., Nowé, A., and Precup, D. (2019b). Per-decision option discounting. In *International Conference on Machine Learning (ICML)*.
- Hasanbeig, M., Jeppu, N. Y., Abate, A., Melham, T., and Kroening, D. (2021). DeepSynth: Automata synthesis for automatic task segmentation in deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*.
- Hengst, B. (2002). Discovering hierarchy in reinforcement learning with HEXQ. In *International Conference on Machine Learning (ICML)*.
- Higgins, I., Matthey, L., Pal, A., Burgess, C. P., Glorot, X., Botvinick, M. M., Mohamed, S., and Lerchner, A. (2016). Beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations (ICLR)*.
- Hinton, G. E. and Zemel, R. (1993). Autoencoders, minimum description length and helmholtz free energy. In *Neural Information Processing Systems (NeurIPS)*.
- Ho, M. K., Abel, D., Griffiths, T. L., and Littman, M. L. (2019). The value of abstraction. *Current opinion in behavioral sciences*, 29:111–116.
- Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Association for Computing Machinery*, 12(10):576–580.

- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9:1735–1780.
- Honkela, A. and Valpola, H. (2004). Variational learning and bits-back coding: An information-theoretic view to bayesian learning. *IEEE Transactions on Neural Networks*, 15(4):800–810.
- Hu, X. and Leung, H.-f. (2023). Provably (more) sample-efficient offline RL with options. In *Neural Information Processing Systems (NeurIPS)*.
- Huang, T., Chen, K., Wei, W., Li, J., Long, Y., and Dou, Q. (2023). Value-informed skill chaining for policy learning of long-horizon tasks with surgical robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Hughes, E., Dennis, M., Parker-Holder, J., Behbahani, F. M. P., Mavalankar, A., Shi, Y., Schaul, T., and Rocktaschel, T. (2024). Open-endedness is essential for artificial superhuman intelligence. In *International Conference on Machine Learning (ICML)*.
- Hung, C.-C., Lillicrap, T. P., Abramson, J., Wu, Y., Mirza, M., Carnevale, F., Ahuja, A., and Wayne, G. (2019). Optimizing agent behavior over long time scales by transporting value. *Nature Communications*, 10:5223.
- Hunt, J. J., Barreto, A., Lillicrap, T. P., and Heess, N. (2018). Entropic policy composition with generalized policy improvement and divergence correction. In *International Conference on Machine Learning (ICML)*.
- Hutsebaut-Buysse, M., Mets, K., and Latré, S. (2022). Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1):172–221.
- Hutter, M. (2014). Extreme state aggregation beyond mdps. In *International Conference on Algorithmic Learning Theory*.
- Icarte, R. T., Klassen, T. Q., Valenzano, R., and McIlraith, S. A. (2022). Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73:173–208.
- Ivanov, A., Bagaria, A., and Konidaris, G. (2025). Discovering options that minimize average planning time. In *AAAI Conference on Artificial Intelligence*.
- Jain, A., Khetarpal, K., and Precup, D. (2021). Safe option-critic: Learning safety in the option-critic architecture. *The Knowledge Engineering Review*, 36:e4.
- James, S. D., Rosman, B., and Konidaris, G. (2020). Learning portable representations for high-level planning. In *International Conference on Machine Learning (ICML)*.
- Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. In *Neural Information Processing Systems (NeurIPS)*.

- Janner, M., Mordatch, I., and Levine, S. (2020). Gamma-models: Generative temporal difference learning for infinite-horizon prediction. In *Neural Information Processing Systems (NeurIPS)*.
- Jaques, N., Lazaridou, A., Hughes, E., Gülçehre, Ç., Ortega, P. A., Strouse, D., Leibo, J. Z., and de Freitas, N. (2019). Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Javed, K. and Sutton, R. S. (2024). The big world hypothesis and its ramifications for artificial intelligence. In *Finding the Frame: An RLC Workshop for Examining Conceptual Frameworks*.
- Jiang, M., Dennis, M., Parker-Holder, J., Foerster, J. N., Grefenstette, E., and Rocktäschel, T. (2021). Replay-guided adversarial environment design. In *Neural Information Processing Systems (NeurIPS)*.
- Jiang, Y., Gu, S. S., Murphy, K. P., and Finn, C. (2019). Language as an abstraction for hierarchical deep reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Jiang, Y., Liu, E. Z., Eysenbach, B., Kolter, J. Z., and Finn, C. (2022). Learning options via compression. In *Neural Information Processing Systems (NeurIPS)*.
- Jiang, Z., Zhang, T., Janner, M., Li, Y., Rocktäschel, T., Grefenstette, E., and Tian, Y. (2023). Efficient planning in a compact latent action space. In *International Conference on Learning Representations (ICLR)*.
- Jinnai, Y., Abel, D., Hershkowitz, D., Littman, M. L., and Konidaris, G. (2019a). Finding options that minimize planning time. In *International Conference on Machine Learning (ICML)*.
- Jinnai, Y., Abel, D., Park, J. W., Hershkowitz, D. E., Littman, M. L., and Konidaris, G. (2019b). Skill discovery with well-defined objectives. In *Structure and Priors in Reinforcement Learning Workshop, International Conference on Learning Representations (ICLR)*.
- Jinnai, Y., Park, J. W., Abel, D., and Konidaris, G. (2019c). Discovering options for exploration by minimizing cover time. In *International Conference on Machine Learning (ICML)*.
- Jinnai, Y., Park, J. W., Machado, M. C., and Konidaris, G. (2020). Exploration in reinforcement learning with deep covering options. In *International Conference on Learning Representations (ICLR)*.
- Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. (2016). The Malmö platform for artificial intelligence experimentation. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

- Jong, N. K., Hester, T., and Stone, P. (2008). The utility of temporal abstraction in reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.
- Jonsson, A. and Barto, A. G. (2006). Causal graph based decomposition of factored MDPs. *Journal of Machine Learning Research*, 7:2259–2301.
- Jothimurugan, K., Alur, R., and Bastani, O. (2019). A composable specification language for reinforcement learning tasks. In *Neural Information Processing Systems (NeurIPS)*.
- Kaelbling, L. P. (1993). Learning to achieve goals. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Kahneman, D. (2011). *Thinking, Fast and Slow*. Farrar, Straus and Giroux.
- Kamat, A. and Precup, D. (2020). Diversity-enriched option-critic. *CoRR*, abs/2011.02565.
- Kanervisto, A., Milani, S., Ramanauskas, K., Topin, N., Lin, Z., Li, J., Shi, J., Ye, D., Fu, Q., Yang, W., Hong, W., Huang, Z., Chen, H., Zeng, G., Lin, Y., Micheli, V., Alonso, E., Fleuret, F., Nikulin, A., and Shpilman, A. (2022). MineRL diamond 2021 competition: Overview, results, and lessons learned. *NeurIPS 2021 Competitions and Demonstrations Track*, pages 13–28.
- Kang, M. and Oh, S. (2022). Deep latent-space sequential skill chaining from incomplete demonstrations. *Intelligent Service Robotics*, 15(2):203–213.
- Kaplan, F. and Oudeyer, P.-Y. (2003). Maximizing learning progress: An internal reward system for development. In *Embodied Artificial Intelligence*.
- Kearns, M. J. and Singh, S. P. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232.
- Khetarpal, K., Ahmed, Z., Comanici, G., Abel, D., and Precup, D. (2020a). What can I do here? A theory of affordances in reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Khetarpal, K., Klissarov, M., Chevalier-Boisvert, M., Bacon, P.-L., and Precup, D. (2020b). Options of interest: Temporal abstraction with interest functions. In *AAAI Conference on Artificial Intelligence*.
- Khetarpal, K., Riemer, M., Rish, I., and Precup, D. (2022). Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476.
- Kim, H. and Mnih, A. (2018). Disentangling by factorising. In *International Conference on Machine Learning (ICML)*.
- Kim, T., Ahn, S., and Bengio, Y. (2019). Variational temporal abstraction. In *Neural Information Processing Systems (NeurIPS)*.

- Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*.
- Kipf, T., Li, Y., Dai, H., Zambaldi, V., Sanchez-Gonzalez, A., Grefenstette, E., Kohli, P., and Battaglia, P. (2019). Compile: Compositional imitation learning and execution. In *International Conference on Machine Learning (ICML)*.
- Klissarov, M., Bacon, P.-L., Harb, J., and Precup, D. (2017). Learning options end-to-end for continuous action tasks. In *Hierarchical Reinforcement Learning Workshop, Neural Information Processing Systems (NeurIPS)*.
- Klissarov, M., D’Oro, P., Sodhani, S., Raileanu, R., Bacon, P.-L., Vincent, P., Zhang, A., and Henaff, M. (2024). Motif: Intrinsic motivation from artificial intelligence feedback. In *International Conference on Learning Representations (ICLR)*.
- Klissarov, M., D’Oro, P., Sodhani, S., Raileanu, R., Bacon, P.-L., Vincent, P., Zhang, A., and Henaff, M. (2025a). MaestroMotif: Skill design from artificial intelligence feedback. In *International Conference on Learning Representations (ICLR)*.
- Klissarov, M., Hjelm, D., Toshev, A. T., and Mazouze, B. (2025b). On the modeling capabilities of large language models for sequential decision making. In *International Conference on Learning Representations (ICLR)*.
- Klissarov, M. and Machado, M. C. (2023). Deep Laplacian-based options for temporally-extended exploration. In *International Conference on Machine Learning (ICML)*.
- Klissarov, M. and Precup, D. (2020). Reward propagation using graph convolutional networks. In *Neural Information Processing Systems (NeurIPS)*.
- Klissarov, M. and Precup, D. (2021). Flexible option learning. In *Neural Information Processing Systems (NeurIPS)*.
- Klyubin, A. S., Polani, D., and Nehaniv, C. L. (2005). All else being equal be empowered. In *Advances in Artificial Life*.
- Kobayashi, S., Schimpf, Y., Schlegel, M., Steger, A., Wolczyk, M., von Oswald, J., Scherrer, N., Maile, K., Lajoie, G., Richards, B. A., Saurous, R., Manyika, J., Arcas, B., Meulemans, A., and Sacramento, J. (2025). Emergent temporal abstractions in autoregressive models enable hierarchical reinforcement learning. *CoRR*, abs/2512.20605.
- Kolve, E., Mottaghi, R., Han, W., Vanderbilt, E., Weihs, L., Herrasti, A., Deitke, M., Ehsani, K., Gordon, D., Zhu, Y., Kembhavi, A., Gupta, A. K., and Farhadi, A. (2017). AI2-THOR: An interactive 3d environment for visual AI. *CoRR*, abs/1712.05474.
- Kompella, V. R., Stollenga, M. F., Luciw, M. D., and Schmidhuber, J. (2017). Continual curiosity-driven skill acquisition from high-dimensional video inputs for humanoid robots. *Artificial Intelligence*, 247:313–335.
- Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. pages 1008–1014.

- Konidaris, G. and Barto, A. (2009a). Efficient skill learning using abstraction selection. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Konidaris, G. and Barto, A. G. (2007). Building portable options: Skill transfer in reinforcement learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Konidaris, G. and Barto, A. G. (2009b). Skill discovery in continuous reinforcement learning domains using skill chaining. In *Neural Information Processing Systems (NeurIPS)*.
- Konidaris, G., Kaelbling, L. P., and Lozano-Perez, T. (2018). From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289.
- Konidaris, G., Kuindersma, S., Grupen, R., and Barto, A. G. (2010). Constructing skill trees for reinforcement learning agents from demonstration trajectories. In *Neural Information Processing Systems (NeurIPS)*.
- Konidaris, G., Kuindersma, S., Grupen, R., and Barto, A. G. (2011). Autonomous skill acquisition on a mobile manipulator. In *AAAI Conference on Artificial Intelligence*.
- Kovač, G., Laversanne-Finot, A., and Oudeyer, P.-Y. (2020). GRIMGEP: Learning progress for robust goal sampling in visual deep reinforcement learning. *IEEE Transactions on Cognitive and Developmental Systems*, 15:1396–1407.
- Krishnan, S., Fox, R., Stoica, I., and Goldberg, K. (2017). DDCO: Discovery of deep continuous options for robot learning from demonstrations. In *Conference on Robot Learning (CoRL)*.
- Kulkarni, T. D., Saedi, A., Gautam, S., and Gershman, S. J. (2016). Deep successor reinforcement learning. *CoRR*, abs/1606.02396.
- Kumar, A., Clune, J., Lehman, J., and Stanley, K. O. (2025a). Questioning representational optimism in deep learning: The fractured entangled representation hypothesis. *CoRR*, abs/2505.11581.
- Kumar, S., Marklund, H., Rao, A., Zhu, Y., Jun Jeon, H., Yueyang, L., and Van Roy, B. (2025b). Continual learning as computationally constrained reinforcement learning. *Foundations and Trends in Machine Learning*, 18(5):913–1053.
- Küttler, H., Nardelli, N., Miller, A. H., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktäschel, T. (2020). The NetHack learning environment. In *Neural Information Processing Systems (NeurIPS)*.
- Kwon, M., Xie, S. M., Bullard, K., and Sadigh, D. (2023). Reward design with language models. In *International Conference on Learning Representations (ICLR)*.
- Kwon, T., Palo, N. D., and Johns, E. (2024). Language models as zero-shot trajectory generators. *IEEE Robotics and Automation Letters*, 9(7):6728–6735.
- Laskin, M., Liu, H., Peng, X. B., Yarats, D., Rajeswaran, A., and Abbeel, P. (2022). CIC: Contrastive intrinsic control for unsupervised skill discovery. *CoRR*, abs/2202.00161.

- LaValle, S. M. (1998). Rapidly-exploring random trees : A new tool for path planning. *The annual research report*.
- Lee, Y., Lim, J. J., Anandkumar, A., and Zhu, Y. (2021). Adversarial skill chaining for long-horizon robot manipulation via terminal state regularization. In *Conference on Robot Learning (CoRL)*.
- Lehnert, L., Larocche, R., and van Seijen, H. (2018). On value function representation of long horizon problems. In *AAAI Conference on Artificial Intelligence*.
- Leibfried, F., Pascual-Díaz, S., and Grau-Moya, J. (2019). A unified Bellman optimality principle combining reward maximization and empowerment. In *Neural Information Processing Systems (NeurIPS)*.
- Létourneau, V., Bellinger, C., Tamblyn, I., and Fraser, M. (2023). Time and temporal abstraction in continual learning: tradeoffs, analogies and regret in an active measuring setting. In *Conference on Lifelong Learning Agents*, pages 470–480. PMLR.
- Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *CoRR*, abs/2005.01643.
- Levy, A., Konidaris, G., Platt, R., and Saenko, K. (2019). Hierarchical reinforcement learning with hindsight. In *International Conference on Learning Representations (ICLR)*.
- Levy, A., Rammohan, S., Allievi, A., Niekum, S., and Konidaris, G. (2023). Hierarchical empowerment: Towards tractable empowerment-based skill learning. *CoRR*, abs/2307.02728.
- Li, H., Yang, X., Wang, Z., Zhu, X., Zhou, J., Qiao, Y., Wang, X., Li, H., Lu, L., and Dai, J. (2024). Auto mc-reward: Automated dense reward design with large language models for Minecraft. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Li, L., Walsh, T. J., and Littman, M. L. (2006). Towards a unified theory of state abstraction for MDPs. *International Symposium on Artificial Intelligence and Mathematics*.
- Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., and Zeng, A. (2023). Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*.
- Lifshitz, S., Paster, K., Chan, H., Ba, J., and McIlraith, S. A. (2023). STEVE-1: A generative model for text-to-behavior in Minecraft. In *Neural Information Processing Systems (NeurIPS)*.
- Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. (2023). Let’s verify step by step. *CoRR*, abs/2305.20050.
- Lin, L.-J. (1991). Self-improvement based on reinforcement learning, planning and teaching. In *Machine Learning Proceedings 1991*, pages 323–327. Morgan Kaufmann.
- Lin, Y.-A., Lee, C.-T., Yang, C.-H., Liu, G.-T., and Sun, S.-H. (2024). Hierarchical programmatic option framework. In *Neural Information Processing Systems (NeurIPS)*.

- Lindemann, S. R. and LaValle, S. M. (2004). Incrementally reducing dispersion by increasing voronoi bias in RRTs. In *IEEE International Conference on Robotics and Automation (ICRA)*.
- Littman, M. L. (2017). The reward hypothesis. Technical report, Brown University. Video presentation can be found at <https://tinyurl.com/4z52r3fe>.
- Liu, G.-T., Hu, E.-P., Cheng, P.-J., Lee, H.-Y., and Sun, S.-H. (2023a). Hierarchical programmatic reinforcement learning via learning to compose programs. In *International Conference on Machine Learning (ICML)*.
- Liu, H., Trott, A., Socher, R., and Xiong, C. (2019). Competitive experience replay. In *International Conference on Learning Representations (ICLR)*.
- Liu, J., Zu, L., He, L., and Wang, D. (2023b). Clue: Calibrated latent guidance for offline reinforcement learning. In *Conference on Robot Learning (CoRL)*.
- Liu, M., Machado, M. C., Tesauro, G., and Campbell, M. (2017). The eigenoption-critic framework. In *Hierarchical Reinforcement Learning Workshop, Neural Information Processing Systems (NeurIPS)*.
- Liu, M., Zhu, M., and Zhang, W. (2022). Goal-conditioned reinforcement learning: Problems and solutions. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Lo, C., Roice, K., Panahi, P. M., Jordan, S. M., White, A., Mihucz, G., Aminmansour, F., and White, M. (2024). Goal-space planning with subgoal models. *Journal of Machine Learning Research*, 25(330):1–57.
- Lobel, S., Bagaria, A., and Konidaris, G. (2023). Flipping coins to estimate pseudocounts for exploration in reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Lozano-Perez, T., Mason, M. T., and Taylor, R. H. (1984). Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, 3(1):3–24.
- Luketina, J., Nardelli, N., Farquhar, G., Foerster, J. N., Andreas, J., Grefenstette, E., Whiteson, S., and Rocktäschel, T. (2019). A survey of reinforcement learning informed by natural language. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Neural Information Processing Systems (NeurIPS)*.
- Luo, Z., Ni, T., Bacon, P.-L., Precup, D., and Si, X. (2025). Understanding behavioral metric learning: A large-scale study on distracting reinforcement learning environments. *Reinforcement Learning Journal*, 6:40–96.
- Ma, C., Ashley, D. R., Wen, J., and Bengio, Y. (2020). Universal successor features for transfer reinforcement learning. *CoRR*, abs/2001.04025.

- Ma, Y. J., Liang, W., Wang, G., Huang, D.-A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., and Anandkumar, A. (2024). Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations (ICLR)*.
- Machado, M. C. (2019). *Efficient Exploration in Reinforcement Learning Through Time-Based Representations*. PhD thesis, University of Alberta, Canada.
- Machado, M. C., Barreto, A., Precup, D., and Bowling, M. H. (2023). Temporal abstraction in reinforcement learning with the successor representation. *Journal of Machine Learning Research*, 24(80):1–69.
- Machado, M. C., Bellemare, M. G., and Bowling, M. H. (2017). A Laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Machado, M. C., Bellemare, M. G., and Bowling, M. H. (2020). Count-based exploration with the successor representation. In *AAAI Conference on Artificial Intelligence*.
- Machado, M. C. and Bowling, M. H. (2016). Learning purposeful behaviour in the absence of rewards. In *Abstraction in Reinforcement Learning Workshop, International Conference on Machine Learning (ICML)*.
- Machado, M. C., Rosenbaum, C., Guo, X., Liu, M., Tesauro, G., and Campbell, M. (2018). Eigenoption discovery through the deep successor representation. In *International Conference on Learning Representations (ICLR)*.
- Mahadevan, S. (2005). Proto-value functions: Developmental reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Mahadevan, S. and Maggioni, M. (2007). Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, pages 2169–2231.
- Mahmoudieh, P., Pathak, D., and Darrell, T. (2022). Zero-shot reward specification via grounded natural language. In *International Conference on Machine Learning (ICML)*.
- Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., and State, G. (2021). Isaac Gym: High performance gpu-based physics simulation for robot learning. In *Neural Information Processing Systems (NeurIPS)*.
- Mann, T. A., Mannor, S., and Precup, D. (2015). Approximate value iteration with temporally extended actions. *Journal of Artificial Intelligence Research*, 53:375–438.
- Mannor, S., Menache, I., Hoze, A., and Klein, U. (2004). Dynamic abstraction in reinforcement learning via clustering. In *International Conference on Machine Learning (ICML)*.
- Mason, M. T. (1985). The mechanics of manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*.

- Massari, F. S., Biehl, M., Meeden, L., and Kanai, R. (2021). Experimental evidence that empowerment may drive exploration in sparse-reward environments. In *International Conference on Development and Learning (ICDL)*.
- Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. (2017). Teacher-student curriculum learning. *IEEE Transactions on Neural Networks and Learning Systems*, 31:3732–3740.
- McClinton, W., Levy, A., and Konidaris, G. (2021). HAC explore: Accelerating exploration with hierarchical reinforcement learning. *CoRR*, abs/2108.02685.
- McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL—the planning domain definition language. Technical report, Yale Center for Computational Vision and Control.
- McGovern, A. and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *International Conference on Machine Learning (ICML)*.
- Medeiros, L. C., Aleixo, D. S., and Lelis, L. H. S. (2022). What can we learn even from the weakest? Learning sketches for programmatic strategies. In *AAAI Conference on Artificial Intelligence*.
- Mees, O., Hermann, L., Rosete-Beas, E., and Burgard, W. (2022). CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*.
- Menache, I., Mannor, S., and Shimkin, N. (2002). Q-Cut—Dynamic discovery of sub-goals in reinforcement learning. In *European Conference on Machine Learning (ECML)*.
- Metzen, J. H. (2012). Online skill discovery using graph-based clustering. In *European Workshop on Reinforcement Learning*.
- Meyerson, A. and Tagiku, B. (2009). Minimizing average shortest path distances via shortcut edge addition. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 272–285. Springer.
- Mishra, U. A., Xue, S., Chen, Y., and Xu, D. (2023). Generative skill chaining: Long-horizon skill planning with diffusion models. In *Conference on Robot Learning (CoRL)*.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518:529–533.
- Mohamed, S. and Rezende, D. J. (2015). Variational information maximisation for intrinsically motivated reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Momennejad, I., Russek, E., Cheong, J. H., Botvinick, M. M., Daw, N. D., and Gershman, S. J. (2017). The successor representation in human reinforcement learning. *Nature Human Behaviour*, 1:680–692.

- Moraes, R. O., Aleixo, D. S., Ferreira, L. N., and Lelis, L. H. S. (2023). Choosing well your opponents: How to guide the synthesis of programmatic strategies. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Moraes, R. O. and Lelis, L. H. S. (2024). Searching for programmatic policies in semantic spaces. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Moraes, R. O., Sadrine, Q. A., Baier, H., and Lelis, L. H. S. (2025). InnateCoder: Learning programmatic options with foundation models. *CoRR*, abs/2505.12508.
- Müller, M. (2007). *Dynamic Time Warping*, pages 69–84. Springer Berlin Heidelberg.
- Mutti, M., Del Col, S., and Restelli, M. (2022). Reward-free policy space compression for reinforcement learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR.
- Nabati, O., Tennenholtz, G., and Mannor, S. (2023). Representation-driven reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Nachum, O., Gu, S. S., Lee, H., and Levine, S. (2018). Data-efficient hierarchical reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Nair, S. and Finn, C. (2020). Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. In *International Conference on Learning Representations (ICLR)*.
- Nam, T., Sun, S., Pertsch, K., Hwang, S. J., and Lim, J. J. (2022). Skill-based meta-reinforcement learning. In *International Conference on Learning Representations (ICLR)*.
- Narayanamurthy, S. M. and Ravindran, B. (2008). On the hardness of finding symmetries in Markov decision processes. In *International Conference on Machine Learning (ICML)*.
- Nayyar, R. K. and Srivastava, S. (2024). Autonomous option invention for continual hierarchical reinforcement learning and planning. *AAAI Conference on Artificial Intelligence*.
- Newman, M. E. J. and Girvan, M. (2004). Finding and evaluating community structure in networks. *American Physical Society*, 69:026113.
- Ni, T., Eysenbach, B., Seyedsalehi, E., Ma, M., Gehring, C., Mahajan, A., and Bacon, P.-L. (2024). Bridging state and history representations: Understanding self-predictive RL. In *International Conference on Learning Representations (ICLR)*.
- Ni, T., Ma, M., Eysenbach, B., and Bacon, P.-L. (2023). When do transformers shine in RL? Decoupling memory from credit assignment. In *Neural Information Processing Systems (NeurIPS)*.
- Oh, J., Hessel, M., Czarnecki, W. M., Xu, Z., van Hasselt, H. P., Singh, S. P., and Silver, D. (2020). Discovering reinforcement learning algorithms. In *Neural Information Processing Systems (NeurIPS)*.

- Open-Ended Learning Team, Stooke, A., Mahajan, A., Barros, C., Deck, C., Bauer, J., Sygnowski, J., Trebacz, M., Jaderberg, M., Mathieu, M., McAleese, N., Bradley-Schmieg, N., Wong, N., Porcel, N., Raileanu, R., Hughes-Fitt, S., Dalibard, V., and Czarnecki, W. M. (2021). Open-ended learning leads to generally capable agents. *CoRR*, abs/2107.12808.
- OpenAI (2024). O1 system card. <https://cdn.openai.com/o1-system-card-20240917.pdf>.
- OpenAI (2025). Computer-using agent. <https://openai.com/index/introducing-operator/>.
- Oudeyer, P. and Kaplan, F. (2007). What is intrinsic motivation? A typology of computational approaches. *Frontiers Neurobotics*, 1:6.
- Palo, N. D., Byravan, A., Hasenclever, L., Wulfmeier, M., Heess, N., and Riedmiller, M. A. (2023). Towards a unified agent with foundation models. In *Reincarnating Reinforcement Learning Workshop, International Conference on Learning Representations (ICLR)*.
- Palo, N. D. and Johns, E. (2024). Keypoint action tokens enable in-context imitation learning in Robotics. *Robotics: Science and Systems Proceedings*.
- Pan, J., Zhang, Y., Tomlin, N., Zhou, Y., Levine, S., and Suhr, A. (2024). Autonomous evaluation and refinement of digital agents. In *Conference on Language Modeling (COLM)*.
- Park, S., Choi, J., Kim, J., Lee, H., and Kim, G. (2022). Lipschitz-constrained unsupervised skill discovery. In *International Conference on Learning Representations (ICLR)*.
- Park, S., Frans, K., Eysenbach, B., and Levine, S. (2025). OGBench: Benchmarking offline goal-conditioned RL. In *International Conference on Learning Representations (ICLR)*.
- Park, S., Ghosh, D., Eysenbach, B., and Levine, S. (2024a). HIQL: Offline goal-conditioned RL with latent states as actions. In *Neural Information Processing Systems (NeurIPS)*.
- Park, S., Kreiman, T., and Levine, S. (2024b). Foundation policies with hilbert representations. In *International Conference on Machine Learning (ICML)*.
- Park, S., Lee, K., Lee, Y., and Abbeel, P. (2023). Controllability-aware unsupervised skill discovery. In *International Conference on Machine Learning (ICML)*.
- Park, S., Rybkin, O., and Levine, S. (2024c). METRA: Scalable unsupervised RL with metric-aware abstraction. In *International Conference on Learning Representations (ICLR)*.
- Parker-Holder, J., Jiang, M., Dennis, M., Samvelyan, M., Foerster, J. N., Grefenstette, E., and Rocktaschel, T. (2022). Evolving curricula with regret-based environment design. *CoRR*, abs/2203.01302.
- Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., and Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Parr, R. and Russell, S. J. (1997). Reinforcement learning with hierarchies of machines. In *Neural Information Processing Systems (NeurIPS)*.

- Pateria, S., Subagdja, B., Tan, A.-H., and Quek, C. (2021). Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys*, 54:1–35.
- Paul, S., Vanbaar, J., and Roy-Chowdhury, A. (2019). Learning from trajectories via subgoal discovery. In *Neural Information Processing Systems (NeurIPS)*.
- Pertsch, K., Lee, Y., and Lim, J. J. (2021). Accelerating reinforcement learning with learned skill priors. In *Conference on Robot Learning (CoRL)*.
- Pfau, D., Petersen, S., Agarwal, A., Barrett, D. G. T., and Stachenfeld, K. L. (2018). Spectral inference networks: Unifying deep and spectral learning. In *International Conference on Learning Representations (ICLR)*.
- Pignatelli, E., Ferret, J., Rocktäschel, T., Grefenstette, E., Paglieri, D., Coward, S., and Toni, L. (2024). Assessing the zero-shot capabilities of LLMs for action evaluation in RL. In *Neural Information Processing Systems (NeurIPS)*.
- Piray, P. and Daw, N. D. (2021). Linear reinforcement learning in planning, grid fields, and cognitive control. *Nature Communications*, 12(1):4942.
- Pitis, S., Chan, H., Zhao, S., Stadie, B. C., and Ba, J. (2020). Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., Kumar, V., and Zaremba, W. (2018). Multi-goal reinforcement learning: Challenging Robotics environments and request for research. *CoRR*, abs/1802.09464.
- Pong, V. H., Dalal, M., Lin, S., Nair, A. V., Bahl, S., and Levine, S. (2020). Skew-Fit: State-covering self-supervised reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Portelas, R., Colas, C., Weng, L., Hofmann, K., and Oudeyer, P.-Y. (2020). Automatic curriculum learning for deep RL: A short survey. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Precup, D. (2000). *Temporal Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst.
- Precup, D., Sutton, R. S., and Singh, S. P. (2000). Eligibility traces for off-policy policy evaluation. In *International Conference on Machine Learning (ICML)*.
- PrismarineJS (2013). PrismarineJS/mineflayer: Create Minecraft bots with a powerful, stable, and high level JavaScript API. <https://github.com/PrismarineJS/mineflayer>.
- Puig, X., Undersander, E., Szot, A., Cote, M. D., Yang, T.-Y., Partsey, R., Desai, R., Clegg, A., Hlavac, M., Min, S. Y., Vondruš, V., Gervet, T., Berges, V.-P., Turner, J. M., Maksymets, O., Kira, Z., Kalakrishnan, M., Malik, J., Chaplot, D. S., Jain, U., Batra, D., Rai, A., and Mottaghi, R. (2024). Habitat 3.0: A co-habitat for humans, avatars, and robots. In *International Conference on Learning Representations (ICLR)*.

- Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.
- Qiu, W. and Zhu, H. (2022). Programmatic reinforcement learning without oracles. In *International Conference on Learning Representations (ICLR)*.
- Racanière, S., Lampinen, A. K., Santoro, A., Reichert, D. P., Firoiu, V., and Lillicrap, T. P. (2019). Automated curricula through setter-solver interactions. In *International Conference on Learning Representations (ICLR)*.
- Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021). Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*.
- Ramesh, R., Tomar, M., and Ravindran, B. (2019). Successor options: An option discovery framework for reinforcement learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Raparthi, S. C., Hambro, E., Kirk, R., Henaff, M., and Raileanu, R. (2023). Generalization to new sequential decision making tasks with in-context learning. *CoRR*, abs/2312.03801.
- Ravindran, B. (2003). SMDP homomorphisms: An algebraic approach to abstraction in semi Markov decision processes. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Ravindran, B. (2004). *An Algebraic Approach to Abstraction in Reinforcement Learning*. University of Massachusetts Amherst.
- Ravindran, B. and Barto, A. G. (2001). Symmetries and model minimization in Markov decision processes.
- Ravindran, B. and Barto, A. G. (2002). Model minimization in hierarchical reinforcement learning. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*.
- Ravindran, B. and Barto, A. G. (2003). Relativized options: Choosing the right transformation. In *International Conference on Machine Learning (ICML)*.
- Ravindran, B. and Barto, A. G. (2004). Approximate homomorphisms: A framework for non-exact minimization in Markov decision processes.
- Rawles, C., Li, A., Rodriguez, D., Riva, O., and Lillicrap, T. P. (2023). AndroidInTheWild: A large-scale dataset for android device control. In *Neural Information Processing Systems (NeurIPS)*.
- Rezaei-Shoshtari, S., Zhao, R., Panangaden, P., Meger, D., and Precup, D. (2022). Continuous MDP homomorphisms and homomorphic policy gradient. In *Neural Information Processing Systems (NeurIPS)*.
- Riemer, M., Cases, I., Rosenbaum, C., Liu, M., and Tesauro, G. (2020). On the role of weight sharing during deep option learning. In *AAAI Conference on Artificial Intelligence*.

- Riemer, M., Liu, M., and Tesauro, G. (2018). Learning abstract options. In *Neural Information Processing Systems (NeurIPS)*.
- Ring, M. B. (1994). *Continual Learning in Reinforcement Environments*. PhD thesis, The University of Texas at Austin.
- Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14(5):465–471.
- Rocamonde, J., Montesinos, V., Nava, E., Perez, E., and Lindner, D. (2024). Vision-language models are zero-shot reward models for reinforcement learning. In *International Conference on Learning Representations (ICLR)*.
- Rodriguez-Sanchez, R., Allen, C. S., and Konidaris, G. (2025). From pixels to factors: Learning independently controllable state variables for reinforcement learning. *CoRR*, abs/2510.02484.
- Rodriguez-Sanchez, R. and Konidaris, G. (2024). Learning abstract world models for value-preserving planning with options. *Reinforcement Learning Journal*, 4:1733–1758.
- Rovee-Collier, C. K. and Gekoski, M. J. (1979). The economics of infancy: A review of conjugate reinforcement. *Advances in child development and behavior*, 13:195–255.
- Salge, C., Glackin, C., and Polani, D. (2014). Empowerment—an introduction. *Guided Self-Organization: Inception*, pages 67–114.
- Salter, S., Wulfmeier, M., Tirumala, D., Heess, N., Riedmiller, M. A., Hadsell, R., and Rao, D. (2022). MO2: Model-based offline options. In *Conference on Lifelong Learning Agents (CoLLAs)*.
- Samvelyan, M., Khan, A., Dennis, M., Jiang, M., Parker-Holder, J., Foerster, J. N., Raileanu, R., and Rocktäschel, T. (2023). MAESTRO: Open-ended environment design for multi-agent reinforcement learning. In *International Conference on Learning Representations (ICLR)*.
- Schaul, T., Borsa, D., Ding, D., Szepesvari, D., Ostrovski, G., Dabney, W., and Osindero, S. (2019). Adapting behaviour for learning progress. *CoRR*, abs/1912.06910.
- Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015). Universal value function approximators. In *International Conference on Machine Learning (ICML)*.
- Schmidhuber, J. (1987). Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook. Master’s thesis, Technische Universität München, Germany.
- Schmidhuber, J. (2004). Optimal ordered problem solver. *Machine Learning*, 54:211–254.
- Schmidhuber, J. (2010). Formal theory of creativity, fun, and intrinsic motivation (1990-2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247.
- Schmidhuber, J. (2011). PowerPlay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in Psychology*, 4.

- Schoknecht, R. (2002). Optimality of reinforcement learning algorithms with linear function approximation. In *Neural Information Processing Systems (NeurIPS)*.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.
- Shapley, L. S. (1953). A value for n-person games. In *Contributions to the Theory of Games II*, pages 307–317. Princeton University Press.
- Sharma, A., Ahn, M., Levine, S., Kumar, V., Hausman, K., and Gu, S. S. (2020a). Emergent real-world robotic skills via unsupervised off-policy reinforcement learning. In *Robotics: Science and Systems (RSS)*.
- Sharma, A., Gu, S. S., Levine, S., Kumar, V., and Hausman, K. (2020b). Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations (ICLR)*.
- Sharma, A., Sharma, M., Rhinehart, N., and Kitani, K. M. (2018). Directed-info GAIL: Learning hierarchical policies from unsegmented demonstrations using directed information. In *International Conference on Learning Representations (ICLR)*.
- Shehmar, D., Schlegel, M., Taylor, M. E., and Machado, M. C. (2026). Laplacian representations for decision-time planning. In *International Conference on Machine Learning (ICML)*.
- Sheth, A. and Roy, K. (2023). Neurosymbolic value-inspired AI (why, what, and how). *IEEE Intelligent Systems*.
- Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905.
- Shrikumar, A., Greenside, P., and Kundaje, A. (2017). Learning important features through propagating activation differences. In *International Conference on Machine Learning (ICML)*.
- Shu, T., Xiong, C., and Socher, R. (2018). Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. In *International Conference on Learning Representations (ICLR)*.
- Silver, D. and Ciosek, K. (2012). Compositional planning using optimal option models. In *International Conference on Machine Learning (ICML)*.
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T. P., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, 550:354–359.
- Silver, D., Singh, S. P., Precup, D., and Sutton, R. S. (2021). Reward is enough. *Artif. Intell.*, 299:103535.

- Silver, T., Dan, S., Srinivas, K., Tenenbaum, J. B., Kaelbling, L. P., and Katz, M. (2024). Generalized planning in pddl domains with pretrained large language models. In *AAAI Conference on Artificial Intelligence*.
- Şimşek, Ö. and Barto, A. G. (2004). Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Şimşek, Ö. and Barto, A. G. (2008). Skill characterization based on betweenness. In *Neural Information Processing Systems (NeurIPS)*.
- Şimşek, Ö., Wolfe, A. P., and Barto, A. G. (2005). Identifying useful subgoals in reinforcement learning by local graph partitioning. In *International Conference on Machine Learning (ICML)*.
- Singh, S. P., Barto, A. G., and Chentanez, N. (2004). Intrinsically motivated reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Singh, S. P., Jaakkola, T., and Jordan, M. (1994). Reinforcement learning with soft state aggregation. In *Neural Information Processing Systems (NeurIPS)*.
- Smith, M., van Hoof, H., and Pineau, J. (2018). An inference-based policy gradient method for learning options. In *International Conference on Machine Learning (ICML)*.
- Sohn, S., Oh, J., and Lee, H. (2018). Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. In *Neural Information Processing Systems (NeurIPS)*.
- Solway, A., Diuk, C., Córdova, N., Yee, D. M., Barto, A. G., Niv, Y., and Botvinick, M. M. (2014). Optimal behavioral hierarchy. *PLOS Computational Biology*, 10(8):1–10.
- Sontakke, S. A., Zhang, J., Arnold, S. M. R., Pertsch, K., Biyik, E., Sadigh, D., Finn, C., and Itti, L. (2023). RoboCLIP: One demonstration is enough to learn robot policies. In *Neural Information Processing Systems (NeurIPS)*.
- Sorg, J. and Singh, S. (2010). Linear options. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.
- Sprekeler, H. (2011). On the relation of slow feature analysis and Laplacian eigenmaps. *Neural Computation*, 23(12):3287–3302.
- Srinivas, A., Krishnamurthy, R., Kumar, P., and Ravindran, B. (2016). Option discovery in hierarchical reinforcement learning using spatio-temporal clustering. *CoRR*, abs/1605.05359.
- Stachenfeld, K. L., Botvinick, M. M., and Gershman, S. J. (2017). The hippocampus as a predictive map. *Nature neuroscience*, 20(11):1643–1653.
- Stolle, M. and Precup, D. (2002). Learning options in reinforcement learning. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*.

- Stout, A. and Barto, A. G. (2010). Competence progress intrinsic motivation. In *International Conference on Development and Learning (ICDL)*.
- Strehl, A. L. and Littman, M. L. (2008). An analysis of model-based interval estimation for Markov decision processes. *J. Comput. Syst. Sci.*, 74(8):1309–1331.
- Strouse, D., Baumli, K., Warde-Farley, D., Mnih, V., and Hansen, S. (2022). Learning more skills through optimistic exploration. In *International Conference on Learning Representations (ICLR)*.
- Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., and Fergus, R. (2018). Intrinsic motivation and automatic curricula via asymmetric self-play. In *International Conference on Learning Representations (ICLR)*.
- Sun, S.-H., Wu, T.-L., and Lim, J. J. (2020). Program guided agent. In *International Conference on Learning Representations (ICLR)*.
- Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44.
- Sutton, R. S. (2019). The bitter lesson. <http://www.incompleteideas.net/IncIdeas/BitterLesson.html>.
- Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.
- Sutton, R. S., Machado, M. C., Holland, G. Z., Szepesvari, D., Timbers, F., Tanner, B., and White, A. (2023). Reward-respecting subtasks for model-based reinforcement learning. *Artificial Intelligence*, 324:104001.
- Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (1999a). Policy gradient methods for reinforcement learning with function approximation. In *Neural Information Processing Systems (NeurIPS)*.
- Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.
- Sutton, R. S., Precup, D., and Singh, S. P. (1998). Intra-option learning about temporally abstract actions. In *International Conference on Machine Learning (ICML)*.
- Sutton, R. S., Precup, D., and Singh, S. P. (1999b). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211.
- Sutton, R. S., Singh, S. P., and McAllester, D. A. (2001). Comparing policy-gradient algorithms. <http://incompleteideas.net/papers/SSM-unpublished.pdf>.

- Sutton, R. S., Szepesvári, C., Geramifard, A., and Bowling, M. (2008). Dyna-style planning with linear function approximation and prioritized sweeping. In *Uncertainty in Artificial Intelligence (UAI)*.
- Szot, A., Clegg, A., Undersander, E., Wijmans, E., Zhao, Y., Turner, J. M., Maestre, N., Mukadam, M., Chaplot, D. S., Maksymets, O., Gokaslan, A., Vondruš, V., Dharur, S., Meier, F., Galuba, W., Chang, A. X., Kira, Z., Koltun, V., Malik, J., Savva, M., and Batra, D. (2021). Habitat 2.0: Training home assistants to rearrange their habitat. In *Neural Information Processing Systems (NeurIPS)*.
- Talvitie, E. (2017). Self-correcting models for model-based reinforcement learning. In *AAAI Conference on Artificial Intelligence*.
- Tang, H., Meng, Z., Hao, J., Chen, C., Graves, D., Li, D., Yu, C., Mao, H., Liu, W., Yang, Y., et al. (2022). What about inputting policy in value function: Policy representation and policy-extended value function approximator. In *AAAI Conference on Artificial Intelligence*.
- Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7).
- Team, S., Raad, M. A., Ahuja, A., Barros, C., Besse, F., Bolt, A., Bolton, A., Brownfield, B., Buttimore, G., Cant, M., Chakera, S., Chan, S. C. Y., Clune, J., Collister, A., Copeman, V., Cullum, A., Dasgupta, I., de Cesare, D., Trapani, J. D., Donchev, Y., Dunleavy, E., Engelcke, M., Faulkner, R., Garcia, F., Gbadamosi, C., Gong, Z., Gonzales, L., Gupta, K., Gregor, K., Hallingstad, A. O., Harley, T., Haves, S., Hill, F., Hirst, E., Hudson, D. A., Hudson, J., Hughes-Fitt, S., Rezende, D. J., Jasarevic, M., Kampis, L., Ke, R., Keck, T., Kim, J., Knagg, O., Kopparapu, K., Lampinen, A., Legg, S., Lerchner, A., Limont, M., Liu, Y., Loks-Thompson, M., Marino, J., Cussons, K. M., Matthey, L., Mccloughlin, S., Mendolicchio, P., Merzic, H., Mitenkova, A., Moufarek, A., Oliveira, V., Oliveira, Y., Openshaw, H., Pan, R., Pappu, A., Platonov, A., Purkiss, O., Reichert, D., Reid, J., Richemond, P. H., Roberts, T., Ruscoe, G., Elias, J. S., Sandars, T., Sawyer, D. P., Scholtes, T., Simmons, G., Slater, D., Soyer, H., Strathmann, H., Stys, P., Tam, A. C., Teplyashin, D., Terzi, T., Vercelli, D., Vujatovic, B., Wainwright, M., Wang, J. X., Wang, Z., Wierstra, D., Williams, D., Wong, N., York, S., and Young, N. (2024). Scaling instructable agents across many simulated worlds. *CoRR*, abs/2404.10179.
- Tedrake, R., Manchester, I. R., Tobenkin, M. M., and Roberts, J. W. (2010). LQR-Trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052.
- Thakoor, S., Rowland, M., Borsa, D., Dabney, W., Munos, R., and Barreto, A. (2022). Generalised policy improvement with geometric policy composition. In *International Conference on Machine Learning (ICML)*.
- Theophilou, A. and Şimşek, Ö. (2025). Novel exploration via orthogonality. In *Neural Information Processing Systems (NeurIPS)*.

- Thomaz, A. L. and Breazeal, C. (2006). Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *AAAI Conference on Artificial Intelligence*.
- Thrun, S. and Pratt, L. (1998). *Learning to Learn: Introduction and Overview*. Kluwer Academic Publishers, USA.
- Thrun, S. and Schwartz, A. (1994). Finding structure in reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Tirinzoni, A., Touati, A., Farebrother, J., Guzek, M., Kanervisto, A., Xu, Y., Lazaric, A., and Pirotta, M. (2025). Zero-shot whole-body humanoid control via behavioral foundation models. In *International Conference on Learning Representations (ICLR)*.
- Todorov, E. (2006). Linearly-solvable Markov decision problems. In *Neural Information Processing Systems (NeurIPS)*.
- Todorov, E. (2009a). Compositionality of optimal control laws. In *Neural Information Processing Systems (NeurIPS)*.
- Todorov, E. (2009b). Efficient computation of optimal actions. *Proceedings of the National Academy of Sciences*, 106(28):11478–11483.
- Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Touati, A. and Ollivier, Y. (2021). Learning one representation to optimize all rewards. In *Neural Information Processing Systems (NeurIPS)*.
- Touati, A., Rapin, J., and Ollivier, Y. (2023). Does zero-shot reinforcement learning exist? In *International Conference on Learning Representations (ICLR)*.
- Trivedi, D., Zhang, J., Sun, S.-H., and Lim, J. J. (2021). Learning to synthesize programs as interpretable and generalizable policies. In *Neural Information Processing Systems (NeurIPS)*.
- Tse, H. T., Chandrasekar, S., and Machado, M. C. (2025). Reward-aware proto-representations in reinforcement learning. *CoRR*, abs/2505.16217.
- Tse, H. T. and Machado, M. C. (2026). DROGO: Default representation objective via graph optimization in reinforcement learning. *CoRR*, abs/2602.00403.
- Uesato, J., Kushman, N., Kumar, R., Song, H. F., Siegel, N. Y., Wang, L., Creswell, A., Irving, G., and Higgins, I. (2023). Solving math word problems with process-based and outcome-based feedback. *CoRR*, abs/2211.14275.
- van Hasselt, H. P. and Wiering, M. A. (2009). Using continuous action spaces to solve discrete problems. In *International Joint Conference on Neural Networks (IJCNN)*.
- Varga, R. S. (2000). *Matrix Iterative Analysis*. Springer.

- Vaswani, A., Shazeer, N. M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Neural Information Processing Systems (NeurIPS)*.
- Vats, S., Likhachev, M., and Kroemer, O. (2023). Efficient recovery learning using model predictive meta-reasoning. In *IEEE International Conference on Robotics and Automation (ICRA)*.
- Veeriah, V., Oh, J., and Singh, S. P. (2018). Many-goals reinforcement learning. *CoRR*, abs/1806.09605.
- Veeriah, V., Zahavy, T., Hessel, M., Xu, Z., Oh, J., Kemaev, I., van Hasselt, H., Silver, D., and Singh, S. P. (2021). Discovery of options via meta-learned subgoals. In *Neural Information Processing Systems (NeurIPS)*.
- Venuto, D., Islam, S. N., Klissarov, M., Precup, D., Yang, S., and Anand, A. (2024). Code as reward: Empowering reinforcement learning with VLMs. In *International Conference on Machine Learning (ICML)*.
- Verma, A., Le, H. M., Yue, Y., and Chaudhuri, S. (2019). Imitation-projected programmatic reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Verma, A., Murali, V., Singh, R., Kohli, P., and Chaudhuri, S. (2018). Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). FeUdal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Victor, B. (2011). Up and down the ladder of abstraction. <https://worrydream.com/LadderOfAbstraction/>.
- Vigorito, C. M. and Barto, A. G. (2008). Autonomous hierarchical skill acquisition in factored MDPs. In *Yale Workshop on Adaptive and Learning Systems*.
- Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T. P., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575:350–354.
- Vlastelica, M., Kolev, P., Cheng, J., and Martius, G. (2023). Diverse offline imitation via fenchel duality. In *European Workshop on Reinforcement Learning*.
- Wan, Y. and Sutton, R. S. (2022). Toward discovering options that achieve faster planning. *CoRR*, abs/2205.12515.

- Wan, Y., Zaheer, M., White, A., White, M., and Sutton, R. S. (2019). Planning with expectation models. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. (2023a). Voyager: An open-ended embodied agent with large language models. *CoRR*, abs/2305.16291.
- Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2016). Learning to reinforcement learn. *CoRR*, abs/1611.05763.
- Wang, K., Zhou, K., Feng, J., Hooi, B., and Wang, X. (2022). Reachability-aware Laplacian representation in reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Wang, K., Zhou, K., Zhang, Q., Shao, J., Hooi, B., and Feng, J. (2021). Towards better Laplacian representation in reinforcement learning with generalized graph drawing. In *International Conference on Machine Learning (ICML)*.
- Wang, T., Torralba, A., Isola, P., and Zhang, A. (2023b). Optimal goal-reaching reinforcement learning via quasimetric learning. In *International Conference on Machine Learning (ICML)*.
- Wang, Y., Sun, Z., Zhang, J., Xian, Z., Biyik, E., Held, D., and Erickson, Z. (2024a). RL-VLM-F: Reinforcement learning from vision language foundation model feedback. In *International Conference on Machine Learning (ICML)*.
- Wang, Y., Xian, Z., Chen, F., Wang, T.-H., Wang, Y., Fragkiadaki, K., Erickson, Z., Held, D., and Gan, C. (2024b). RoboGen: Towards unleashing infinite data for automated robot learning via generative simulation. In *International Conference on Machine Learning (ICML)*.
- Wang, Z., Cai, S., Liu, A., Jin, Y., Hou, J., Zhang, B., Lin, H., He, Z., Zheng, Z., Yang, Y., Ma, X., and Liang, Y. (2023c). Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *arXiv preprint arXiv: 2311.05997*.
- Wang, Z. Z., Mao, J., Fried, D., and Neubig, G. (2024c). Agent workflow memory. *CoRR*, abs/2409.07429.
- Warde-Farley, D., de Wiele, T. V., Kulkarni, T. D., Ionescu, C., Hansen, S., and Mnih, V. (2019). Unsupervised control through non-parametric discriminative rewards. In *International Conference on Learning Representations (ICLR)*.
- Watkins, C. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q. V., and Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. In *Neural Information Processing Systems (NeurIPS)*.

- Wen, Z., Precup, D., Ibrahimi, M., Barreto, A., Van Roy, B., and Singh, S. (2020). On efficiency in hierarchical reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Wilkes, M. V., Wheeler, D. J., and Gill, S. (1951). *The Preparation of Programs for an Electronic Digital Computer*. Addison-Wesley Press.
- Wirth, C., Akrou, R., Neumann, G., and Fürnkranz, J. (2017). A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 18(136):1–46.
- Wiskott, L. and Sejnowski, T. J. (2002). Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770.
- Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.
- Wu, Y., Tucker, G., and Nachum, O. (2019). The Laplacian in RL: learning representations with efficient approximations. In *International Conference on Learning Representations (ICLR)*.
- Wulfmeier, M., Rao, D., Hafner, R., Lampe, T., Abdolmaleki, A., Hertweck, T., Neunert, M., Tirumala, D., Siegel, N. Y., Heess, N., and Riedmiller, M. A. (2020). Data-efficient hindsight off-policy option learning. In *International Conference on Machine Learning (ICML)*.
- Xiao, T., Chan, H., Sermanet, P., Wahid, A., Brohan, A., Hausman, K., Levine, S., and Tompson, J. (2022). Robotic skill acquisition via instruction augmentation with vision-language models. *Robotics: Science and Systems Proceedings*.
- Xie, T., Zhao, S., Wu, C. H., Liu, Y., Luo, Q., Zhong, V., Yang, Y., and Yu, T. (2024). Text2Reward: Automated dense reward function generation for reinforcement learning. In *International Conference on Learning Representations (ICLR)*.
- Xie, Z., Ji, C., and Zhang, Y. (2022). Deep skill chaining with diversity for multi-agent systems. In *International Conference on Artificial Intelligence (ICAI)*.
- Xu, Z., van Hasselt, H. P., and Silver, D. (2018). Meta-gradient reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Yang, R., Chen, J., Zhang, Y., Yuan, S., Chen, A., Richardson, K., Xiao, Y., and Yang, D. (2025). SelfGoal: Your language agents already know how to achieve high-level goals. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.
- Yang, R., Fang, M., Han, L., Du, Y., Luo, F., and Li, X. (2021a). MHER: Model-based hindsight experience replay. In *Deep Reinforcement Learning Workshop, Neural Information Processing Systems (NeurIPS)*.
- Yang, Y., Inala, J. P., Bastani, O., Pu, Y., Solar-Lezama, A., and Rinard, M. (2021b). Program synthesis guided reinforcement learning for partially observed environments. In *Neural Information Processing Systems (NeurIPS)*.

- Yu, W., Gileadi, N., Fu, C., Kirmani, S., Lee, K.-H., Arenas, M. G., Chiang, H.-T. L., Erez, T., Hasenclever, L., Humprik, J., Ichter, B., Xiao, T., Xu, P., Zeng, A., Zhang, T., Heess, N., Sadigh, D., Tan, J., Tassa, Y., and Xia, F. (2023). Language to rewards for robotic skill synthesis. In *Conference on Robot Learning (CoRL)*.
- Zahavy, T., Haroush, M., Merlis, N., Mankowitz, D. J., and Mannor, S. (2018). Learn what not to learn: Action elimination with deep reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.
- Zahavy, T., O’Donoghue, B., Barreto, A., Flennerhag, S., Mnih, V., and Singh, S. P. (2021). Discovering diverse nearly optimal policies with successor features. In *Unsupervised Reinforcement Learning Workshop, International Conference on Machine Learning (ICML)*.
- Zahavy, T., Schroecker, Y., Behbahani, F. M. P., Baumli, K., Flennerhag, S., Hou, S., and Singh, S. P. (2022). Discovering policies with DOMiNO: Diversity optimization maintaining near optimality. In *International Conference on Learning Representations (ICLR)*.
- Zala, A., Cho, J., Lin, H., Yoon, J., and Bansal, M. (2024). EnvGen: Generating and adapting environments via LLMs for training embodied agents. In *Conference on Language Modeling (COLM)*.
- Zhang, A., McAllister, R., Calandra, R., Gal, Y., and Levine, S. (2021a). Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations (ICLR)*.
- Zhang, J., Lehman, J., Stanley, K. O., and Clune, J. (2024). OMNI: Open-endedness via models of human notions of interestingness. In *International Conference on Learning Representations (ICLR)*.
- Zhang, J., Pertsch, K., Yang, J., and Lim, J. J. (2021b). Minimum description length skills for accelerated reinforcement learning. In *Self-Supervision for Reinforcement Learning Workshop, International Conference on Learning Representations (ICLR)*.
- Zhang, M., Tang, H., Hao, J., and Zheng, Y. (2023). Towards a unified policy abstraction theory and representation learning approach in Markov decision processes. In *International Conference on Learning Representations (ICLR)*.
- Zhang, S. and Whiteson, S. (2019). DAC: The double actor-critic architecture for learning options. In *Neural Information Processing Systems (NeurIPS)*.
- Zhang, Y., Abbeel, P., and Pinto, L. (2020). Automatic curriculum learning through value disagreement. In *Neural Information Processing Systems (NeurIPS)*.
- Zhao, Z., Samel, K., Chen, B., and Song, L. (2021). ProTo: Program-guided transformer for program-guided tasks. In *Neural Information Processing Systems (NeurIPS)*.
- Zheng, Q., Henaff, M., Zhang, A., Grover, A., and Amos, B. (2024). Online intrinsic rewards for decision making agents from large language model feedback. *CoRR*, abs/2410.23022.

Zhou, S., Xu, F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Bisk, Y., Fried, D., Alon, U., and Neubig, G. (2024). WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*.