Autonomous Invention of Generalizable Options for Hierarchical Planning and Reinforcement Learning

Rashmeet Kaur Nayyar and **Siddharth Srivastava**

Autonomous Agents and Intelligent Robots Lab,
School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ, USA
{rmnayyar, siddharths}@asu.edu

Abstract

Abstraction is key to scaling up reinforcement learning (RL). However, autonomously learning abstract state and action representations to enable transfer and generalization remains a challenging open problem. This paper presents a novel approach for inventing, representing, and utilizing options, which represent temporally extended behaviors, in continual RL settings. Our approach addresses streams of stochastic problems characterized by long horizons, sparse rewards, and unknown transition and reward functions.

Our approach continually learns and maintains an interpretable state abstraction, and uses it to invent high-level options with abstract symbolic representations. These options meet three key desiderata: (1) composability for solving tasks effectively with lookahead planning, (2) reusability across problem instances for minimizing the need for relearning, and (3) mutual independence for reducing interference among options. Our main contributions are approaches for continually learning transferable, generalizable options with symbolic representations, and for integrating search techniques with RL to efficiently plan over these learned options to solve new problems. Empirical results demonstrate that the resulting approach effectively learns and transfers abstract knowledge across problem instances, achieving superior sample efficiency compared to state-of-the-art methods.

1 Introduction

Generalization in Reinforcement Learning (RL) for enabling efficient autonomous decision-making has been constrained by two fundamental challenges: sample inefficiency and poor scalability, particularly in environments with long horizons and sparse rewards. To address these limitations, researchers have focused on: (1) state abstraction, which creates compact state representations (Jong and Stone 2005; Dadvar, Nayyar, and Srivastava 2023), and (2) temporal abstraction, which captures hierarchical task structures through temporally extended behaviors (Barto and Mahadevan 2003; Pateria et al. 2021), such as options (Sutton, Precup, and Singh 1999). Abstraction-based methodologies offer principled approaches for knowledge transfer across tasks (Abel et al. 2018), especially in the challenging setting of continual learning (Liu, Xiao, and Stone 2021; Khetarpal et al. 2022), where agents must interact with and solve tasks indefinitely. However, most existing research on option discovery in RL focuses either on continuous control tasks with short horizons and dense rewards (Bagaria, Senthil, and Konidaris 2021; Klissarov and Precup 2021), on single-task settings (Bagaria and Konidaris 2020; Riemer, Liu, and Tesauro 2018), or lacks support for lookahead planning over options to guide low-level policy learning (Machado et al. 2017; Khetarpal et al. 2020). A critical challenge remains open: the autonomous discovery of generalizable, reusable options for long-horizon, sparse-reward tasks in continual RL settings. This is particularly relevant to real-world scenarios such as warehouse management, disaster recovery operations, and assembly tasks, where agents must adapt to shifts in context and required behaviors without dense reward feedback and closed-form analytical models.

We present a novel approach that coherently addresses option discovery and transfer with a unified symbolic abstraction framework for factored domains in the continual RL settings. We focus on long-horizon, goal-based Markov decision processes (MDPs) in RL settings with unknown transition functions and sparse rewards. In particular, we consider three conceptual desiderata for options: 1) *composability*: support chaining options for enabling hierarchical planning, 2) *reusability*: support transfer of options to new problem instances, minimizing the need for relearning, and 3) *mutual independence*: reduce interference among options, allowing options to be learned and executed independently with minimal side effects while ensuring composability at well-defined endpoints. Most prior works meet some of these criteria but not all.

Our approach, **C**ontinual **Hi**erarchical **R**einforcement Learning and **P**lanning (CHiRP) takes as input a set of state variables and a stochastic simulator, and invents options satisfying desiderata 1-3: the invented options have symbolic abstract descriptions that directly support composability and reusability through high-level planning; these options have stronger effects on different sets of variables and/or values, supporting mutual independence. The core idea is to capture notions of context-specific abstractions that depend on and change with the current state by identifying salient variable values responsible for greatest variation in the Q-function, and to use changes in these abstractions as a cue for defining option endpoints. With every new task in a continual stream

of problems, CHiRP transfers these options and invents new options, building a model of options that is more broadly useful (Fig. 1). For example, in large instances of the well-known taxi domain (Dietterich 2000), CHiRP autonomously invents four key options: navigate to the passenger location, pickup the passenger, navigate to the dropoff location, and dropoff the passenger.

Extensive empirical evaluation across a variety of challenging domains with continuous/hybrid states and discrete actions demonstrates that our approach substantially surpasses SOTA RL baselines in sample efficiency within continual RL settings. Key strengths of our approach are: fewer hyper-parameters and less tuning required compared to many of the baselines, including SOTA DRL methods that require extensive architecture tuning, greater interpretability, increased sample-efficiency, and satisfaction of key conceptual desiderata for task decomposition.

To the best of our knowledge, CHiRP is the first approach to autonomously invent composable, reusable, and mutually independent options using auto-generated state abstractions, and to use these options to create a novel hierarchical paradigm for continual RL in long-horizon, sparse reward settings. Our main contributions are: (a) a novel approach for auto-inventing symbolic options with abstract representations, (b) a novel search process for composing options for solving new tasks, and (c) a hierarchical framework that integrates planning and learning for continual RL.

2 Formal Framework

Problem Definition. We assume RL settings where an agent interacts with a goal-oriented Markov decision process (MDP) $\mathcal{M}$ defined by the combination of an environment $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, h \rangle$ and a task $\langle s_i, S_g, \mathcal{R} \rangle$. Here, $\mathcal{S}$ is a set of states defined using a factored representation, where $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a set of continuous real-valued or discrete variables, and each $v_i \in \mathcal{V}$ has an ordered domain $\mathcal{D}_{v_i} = [\mathcal{D}^{min}_{v_i}, \mathcal{D}^{max}_{v_i})$. We denote the value of variable v_i in state s as $s(v_i)$. Each state $s \in \mathcal{S}$ is defined by assigning each $v_i \in \mathcal{V}$ a value from its domain, i.e., $s(v_i) \in \mathcal{D}_{v_i}$. $\mathcal{A}$ is a set of finite actions. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mu\mathcal{S}$ is a stochastic transition function where $\mu\mathcal{S}$ is a probability measure on $\mathcal{S}$. $\gamma \in (0, 1]$ is a discount factor and h is a horizon. Lastly, $s_i \in \mathcal{S}$ is an initial state, $S_g \subseteq \mathcal{S}$ is a set of goal states, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a reward function underlying the task.

Running Example. Consider a hybrid state space (defined by both continuous and finite variables) adaptation of the classic taxi domain (Dietterich 1999), where a taxi starts at a random location and is tasked with picking up a passenger and transporting them to their destination. The pickup and dropoff locations are chosen randomly among n specific locations. States are represented by variables $\mathcal{V}$: $x \in \mathbb{R}$ (taxi's x-coordinate), $y \in \mathbb{R}$ (taxi's y-coordinate), $l \in \{0, 1, \ldots, n\}$ (passenger's location, where integers indicate specific locations and 0 indicates elsewhere), and $p \in \{0, 1\}$ (passenger's presence in the taxi). For clarity, consider variables with small domains: $\mathcal{D}_x = [0.0, 5.0), \mathcal{D}_y = [0.0, 5.0)$, $\mathcal{D}_l = \{0, 1, 2, 3, 4\}$, and $\mathcal{D}_p = \{0, 1\}$. A state assigns a value to each variable from its domain, e.g., $s = \langle s(x) = 0.9, s(y) = 2.1, s(l) = 3, s(p) = 0 \rangle$. There are six primitive actions: navigate in four cardinal directions by a fixed distance, pickup the passenger, and drop-off the passenger.

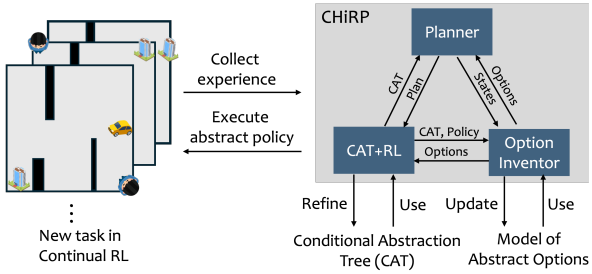

Figure 1: Overall approach for Continual Hierarchical Reinforcement Learning and Planning (CHiRP).

A solution to a goal-oriented MDP is a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, which maps each state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$, with the objective of maximizing the expected discounted cumulative reward. When analytical models of transition $\mathcal{T}$ and reward $\mathcal{R}$ functions are available, classical dynamic programming methods such as value iteration (Bellman 1957) and policy iteration (Howard 1960) are used to compute policies. However, in RL settings, $\mathcal{T}(\mathcal{S}, \mathcal{A})$ and $\mathcal{R}(\mathcal{S}, \mathcal{A})$ can be sampled but their closed-form analytical models are not available. Methods like Q-learning (Watkins and Dayan 1992), DQN (Mnih et al. 2013), and PPO (Schulman et al. 2017) are designed to learn policies directly from samples, but they are often sample inefficient and struggle to scale when effective horizons are long (Laidlaw, Russell, and Dragan 2023) and rewards are sparse (Dadvar, Nayyar, and Srivastava 2023).

Continual Reinforcement Learning. Many challenging real-world scenarios are captured by continual or lifelong learning setting (Ring 1994; Liu, Xiao, and Stone 2021), where an agent must interact with and solve a stream of related tasks, randomly sampled from a distribution, over the course of its lifetime. In these tasks, subtle aspects of the initial state, goal states, transition function, and reward function change over time. The goal is to efficiently retain and reuse knowledge from previous experiences to solve new tasks. We adapt the definition of continual learning (Khetarpal et al. 2022) to goal-oriented MDPs as follows.

Definition 2.1 (Continual Reinforcement Learning (CRL))**.** CRL problem is a stream of n MDPs $\boldsymbol{\mathcal{M}}$ where each MDP $\mathcal{M} \in \boldsymbol{\mathcal{M}}$ shares $\langle \mathcal{S}, \mathcal{A}, \gamma, h \rangle$ and may have distinct $\langle \mathcal{T}, s^{\mathcal{M}}_i, S^{\mathcal{M}}_g, \mathcal{R}^{\mathcal{M}} \rangle$. An agent interacts with each $\mathcal{M}_i \in \boldsymbol{\mathcal{M}}$ for a maximum of $\mathcal{H}$ timesteps in the order $i = 1, \ldots, n$.

A solution to a CRL problem is a policy $\pi^{\mathcal{M}} : \mathcal{S} \rightarrow \mathcal{A}$ for each MDP $\mathcal{M}$ in the problem stream $\boldsymbol{\mathcal{M}}$. The goal is to solve each $\mathcal{M} \in \boldsymbol{\mathcal{M}}$ while minimizing agent interactions and maximizing the expected discounted cumulative reward. In such challenging settings, abstraction techniques emerge as powerful tools for improving scalability and generalization in RL (Li, Walsh, and Littman 2006).

State Abstractions. A state abstraction $\phi : \mathcal{S} \rightarrow \overline{\mathcal{S}}$ maps each state $s \in \mathcal{S}$ to an abstract state $\overline{s} \in \overline{\mathcal{S}}$, where $\overline{\mathcal{S}}$ is a par-

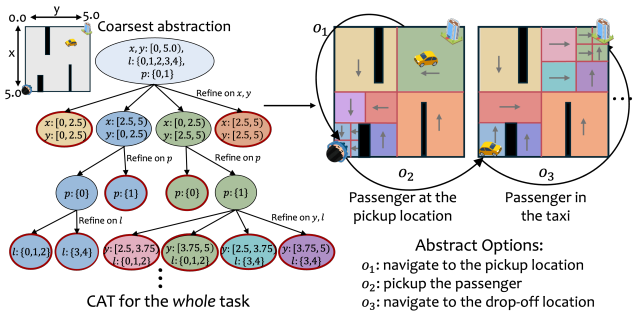

Figure 2: Illustration of a Conditional Abstraction Tree (CAT) (left) and Abstract Options (right) for a small instance in taxi world. Left: Nodes show values of refined variables; other variables inherit values from parent nodes. Right: Arrows denote option policies. Abstract states are highlighted with solid red lines in both figures.

titioning of $\mathcal{S}$. Given a set of variables $\mathcal{V}$, let $\overline{s}(v_i)$ denote the value of $v_i \in \mathcal{V}$ in an abstract state $\overline{s}$. An abstract state assigns an interval of values to each variable from its domain, e.g., state $s = \langle s(x) = 2.6, s(y) = 0.9, s(l) = 3, s(p) = 0\rangle$ can be abstracted as $\overline{s} = \langle \overline{s}(x) = [2.5, 5), \overline{s}(y) = [0.0, 2.5), \overline{s}(l) = \{3, 4\}, \overline{s}(p) = \{0\}\rangle$. Here, $\overline{s}$ represents a set of states $\{s \in \mathcal{S} | \forall v_i \in \mathcal{V},\ s(v_i) \in \overline{s}(v_i)\}$. Also, the coarsest state abstraction contains a single abstract state $\overline{s}_{init}$ that assigns $\overline{s}_{init}(v_i) = \mathcal{D}_{v_i}$ to each $v_i \in \mathcal{V}$. Formally, an abstract state is defined as follows.

Definition 2.2 (Abstract State)**.** Given a set of variables $\mathcal{V}$ and the domain $\mathcal{D}_{v_i}$ for each variable $v_i \in \mathcal{V}$, an *abstract state* $\overline{s} \in \overline{\mathcal{S}}$ is defined by assigning an interval of values $\overline{s}(v_i) \subseteq \mathcal{D}_{v_i}$ to each $v_i \in \mathcal{V}$.

Conditional Abstraction Trees (CATs). State abstraction on a variable's values (such as taxi's location) is *conditioned* on the values of the other variables (such as passenger's presence in the taxi). Such rich conditional abstractions for a task can be captured in the form of a Conditional Abstraction Tree (CAT) (Dadvar, Nayyar, and Srivastava 2023) where the root denotes the coarsest abstract state, while lower-level nodes represent abstract states with greater refinement on variables requiring higher resolution in variable values. Fig. 2 illustrates a CAT for a small problem of taxi domain. CATs are defined formally as follows.

Definition 2.3 (Conditional Abstraction Trees (CATs))**.** A *CAT* Δ is a tuple $\langle \mathcal{N}, \mathcal{E}\rangle$, where $\mathcal{N}$ is a set of nodes representing possible abstract states and $\mathcal{E}$ is a set of directed edges connecting these nodes. The root represents the coarsest abstract state $\overline{s}_{init}$. An edge $e \in \mathcal{E}$ from a parent abstract state $\overline{s}_p \in \mathcal{N}$ to a child abstract state $\overline{s}_c \in \mathcal{N}$ exists iff $\overline{s}_c$ can be obtained by splitting atleast one of the variable intervals in $\overline{s}_p$ at most once. The leaf nodes represent the active abstract state space $\overline{\mathcal{S}}_\Delta$. Δ defines a state abstraction $\phi_\Delta : \mathcal{S} \to \overline{\mathcal{S}}_\Delta$ mapping each state $s \in \mathcal{S}$ to the abstract state $\overline{s} \in \overline{\mathcal{S}}_\Delta$ represented by the unique leaf in Δ containing s.

In this work, we use novel notions based on CAT-based state abstractions to coherently address option invention and transfer with a unified abstraction framework. We compute CATs online using CAT+RL (Dadvar, Nayyar, and Srivastava 2023) which refines states with high dispersion in TD-errors during Q-learning over the abstract state space.

Abstract Options. We use the standard notion of options (Sutton, Precup, and Singh 1999). An option o is a triple $\langle \mathcal{I}_o, \beta_o, \pi_o\rangle$, where $\mathcal{I}_o \subset \mathcal{S}$ is the initiation set where o can initiate, $\beta_o \subset \mathcal{S}$ is the termination set where o terminates, and $\pi_o : \mathcal{S} \to \mathcal{A}$ is the option policy prescribed by o that maps states to actions. Our approach autonomously learns all components of options, defined over an abstract state space $\overline{\mathcal{S}}_{\Delta_o}$ as follows. We define an *abstract option* o as a tuple $\langle \Delta_o, \mathcal{I}_o, \beta_o, \pi_o\rangle$, where Δ_o is the CAT-based state abstraction $\phi_{\Delta_o} : \mathcal{S} \to \overline{\mathcal{S}}_{\Delta_o}$, $\mathcal{I}_o \subset \overline{\mathcal{S}}_{\Delta_o}$ is the abstract initiation set, $\beta_o \subset \overline{\mathcal{S}}_{\Delta_o}$ is the abstract termination set, and $\pi_o : \overline{\mathcal{S}}_{\Delta_o} \to \mathcal{A}$ is the abstract partial policy. $\mathcal{I}_o$ and β_o denote *option endpoints*. The declarative description of an option is termed as *option signature* $\langle \mathcal{I}_o, \beta_o\rangle$. Additionally, two options o_i and o_j are composable iff $\beta_{o_i} \subseteq \mathcal{I}_{o_j}$.

3 Continual Hierarchical RL and Planning

The core contribution of this paper is a novel approach for autonomously inventing a forward model of abstract options using auto-generated CAT-based state abstractions and efficiently utilizing them for solving continual RL problems. Our approach CHiRP (Fig. 1) takes as input a continual stream of tasks $\boldsymbol{\mathcal{M}}$ and a stochastic simulator, and computes a policy for each task. The key insight for option invention is that CATs, auto-generated using CAT+RL, for each task inherently capture abstractions that remain stable within a subtask, but change significantly across subtasks within the task. We use CATs to capture notions of context-specific abstractions that depend on the current state and then use changes in these salient abstractions as a cue for defining option endpoints. For instance, in the taxi domain (Fig. 2), when the passenger has not been picked up, the abstraction needs greater refinement on the value of the taxi's location closer to the passenger's location. However, when the context changes to a situation where the passenger is in the taxi and has not been dropped off, the abstraction needs greater refinement on the value of the taxi's location near the destination. In this scenario, the pickup option (option o_2 in Fig. 2) can be seen as an option that achieves a significant change in context-specific abstractions.

CHiRP maintains a universal CAT created from the current and all previous problems in the stream, and exploits its structure to identify context-variables, such as the passenger's presence in the taxi. These variables are used to define a context-specific distance between states in a manner such that higher distances correspond to greater changes in salient variables and values. CHiRP operationalizes this notion of changes in saliency to invent abstract options. Note that the descriptions of the invented options are symbolic, hence directly support efficient composition and reuse. When a new task is encountered, CHiRP uses foresight to plan ahead by connecting endpoints of learned options, while also inventing additional option signatures to bridge gaps if needed. Each option maintains its own encapsulated CAT-based state abstraction, allowing options to be used and updated inde-

pendently. The options have stronger effects on different sets of variables and values, which reduces mutual interference.

In Sec. 3.1, we present CHiRP, our overall approach to continual RL through autonomous invention, transfer, and reuse of abstract options. Sec. 3.2 details our novel approach for option invention, and Sec. 3.3 explains a novel planner for composing these options to solve new tasks.

3.1 Algorithm Overview

Given a continual RL problem, Alg. 1 begins with an empty model of abstract options $\mathcal{O}$ and a CAT Δ with the coarsest state abstraction (lines 1-2). For each new task in the stream, the CAT's abstraction is used to compute the initial and goal abstract states (line 4). Once a solution policy is found or a budget of $\mathcal{H}$ timesteps is reached, the agent moves to solving the next task (line 5). To solve the current task, the agent interleaves: (1) a planner to plan with the current model of abstract options, (2) CAT+RL to refine the current CAT's state abstraction during learning, and (3) an option inventor to invent novel abstract options using the updated CAT (lines 6-16). The updated model of options $\mathcal{O}$ and CAT Δ are transferred to solve subsequent tasks (line 3).

Given a new task, Alg. 1 uses an offline search process with the current model of abstract options to compute a plan from the current abstract state to a goal abstract state, denoted by $\Pi = \langle o_0, \ldots, o_n \rangle$, $o_i \in \mathcal{O}$. The learned option representations are used to compose this plan, as detailed in Sec. 3.3 (line 6 computeOptionPlan()). The method additionally creates new option signatures to allow connecting endpoints of learned options with gaps between them. If no plan is found with the current model, we initialize the plan with a new option signature from the current abstract state to the goal abstract states (line 8). The CAT and policies for these options are learned later during RL.

For each newly created option signature or previously learned option in the plan, $o \in \Pi$, we generate an MDP with a sparse intrinsic reward for reaching the option's termination (line 10). The option's CAT Δ_o and policy π_o are then learned or fine-tuned using CAT+RL (line 11). We use these option-specific CATs and policies to invent new abstract options with updated representations, as detailed in Sec. 3.2 (line 13 inventOptions()). This process converts option signatures into abstract options with learned CATs and policies. We also update the universal CAT Δ with each invented option's CAT Δ_o, adjusting the current abstract state (line 14). Note that option executions can be stochastic, and the agent may fail to successfully reach the termination set of an option. In such cases, we use active replanning (Kaelbling and Lozano-Pérez 2011) from the current abstract state to a goal abstract state (line 16), and continue learning option-specific CATs and policies. This process repeats until the computed plan of abstract options successfully solves the problem. Finally, Alg. 1 transfers the updated model $\mathcal{O}$ with the new options and CAT Δ to solve new tasks.

3.2 Inventing Generalizable Options

We now discuss our approach for inventing options using a learned CAT and an abstract policy (Alg. 1 line 13) (Sec. 3.1 explains our approach for obtaining these inputs). The key idea is to identify transitions that lead to significant changes in context-specific abstractions, revealing that the nature of the task has changed. We recognize changes in sets of salient variables and values that significantly impact changes in the CAT's structure, and use this as an indicator for determining when to initiate and terminate options. We also exploit the structure of different subtrees in the CAT to further decompose these options into multiple options.

We first describe our overall approach for option invention (Alg. 2) and then discuss each component in detail. Alg. 2 uses the input abstract policy π to compute a roll-out trajectory $\tau = \langle s_0, \ldots, s_n \rangle$ and then uses the input CAT's abstraction function to compute its corresponding abstract trajectory $\overline{\tau}$ (lines 1-2). Context-specific abstractions are generated for each state in the trajectory (line 3) and used to identify a sequence of option endpoints τ^* (line 4). We use these identified option endpoints to invent abstract options as follows. For each consecutive option endpoints $s_i^*, s_j^* \in \tau^*$, we first compute a segment $\text{seg}_{ij} = \langle \bar{s}_k, \ldots, \bar{s}_m \rangle \subseteq \overline{\tau}$ s.t. $\bar{s}_k = s_i^*$ and $\bar{s}_{m+1} = s_j^*$. Then, an abstract option $o_{ij} = \langle \Delta_{o_{ij}}, \mathcal{I}_{o_{ij}}, \beta_{o_{ij}}, \pi_{o_{ij}} \rangle$ is invented, where $\mathcal{I}_{o_{ij}}$ contains siblings of s_i^* in the CAT Δ that are in seg_{ij}, $\beta_{o_{ij}} = \{s_j^*\}$,

Algorithm 1: CHiRP algorithm

Input: Stream of MDPs $\boldsymbol{\mathcal{M}}$, Budget $\mathcal{H}$
Output: Policy $\pi^{\mathcal{M}}$ for each $\mathcal{M} \in \boldsymbol{\mathcal{M}}$

1 $\mathcal{O} \leftarrow$ Initialize empty model of abstract options
2 $\Delta \leftarrow$ Initialize CAT with $\overline{s}_{init}$
3 **for** $\mathcal{M} \in \boldsymbol{\mathcal{M}}$ **do**
4 $\quad s \leftarrow s_i^{\mathcal{M}}$; $\overline{s}$, $\overline{S}_g \leftarrow$ abstractStates(Δ, $s_i^{\mathcal{M}}$, $S_g^{\mathcal{M}}$)
5 $\quad$ **while** $\pi^{\mathcal{M}}$ *not found or steps* $< \mathcal{H}$ **do**
6 $\quad\quad \Pi \leftarrow$ computeOptionPlan($\mathcal{O}$, Δ, $\overline{s}$, $\overline{S}_g$)
7 $\quad\quad$ **if** Π *is not found* **then**
8 $\quad\quad\quad \Pi \leftarrow$ inventOptionSign(s, $S_g^{\mathcal{M}}$)
9 $\quad\quad$ **for** $o \in \Pi$ **do**
10 $\quad\quad\quad \mathcal{M}_o \leftarrow$ generate MDP for o
11 $\quad\quad\quad \Delta_o, \pi_o \leftarrow$ CAT+RL($\mathcal{M}_o$, Δ, s)
12 $\quad\quad\quad$ **if** π_o *is learned* **then**
13 $\quad\quad\quad\quad \mathcal{O}$.update(inventOptions($\Delta_o$,$\pi_o$))
14 $\quad\quad\quad\quad \Delta \leftarrow \Delta_o$; update $s, \overline{s}$
15 $\quad\quad\quad$ **else**
16 $\quad\quad\quad\quad$ break and replan Π
17 **return** $\forall \mathcal{M} \in \boldsymbol{\mathcal{M}}\ \pi^{\mathcal{M}}$

Algorithm 2: Invention of Abstract Options

Input: CAT Δ, Policy π, thresholds δ_{thre} and σ_{thre}
Output: Abstract options $\mathcal{O}$

1 $\tau \leftarrow$ computeTrajectory(π)
2 $\overline{\tau} \leftarrow$ computeAbstractTrajectory(τ,Δ)
3 $\Delta_\tau \leftarrow$ generateContext-SpecificCATs(Δ,τ)
4 $\tau^* \leftarrow$ identifyOptionEndPoints(Δ,Δ_τ,$\overline{\tau}$,δ_{thre},σ_{thre})
5 $\mathcal{O} \leftarrow$ inventAbstractOptions(τ^*,$\overline{\tau}$,Δ,π)
6 $\mathcal{O} \leftarrow$ finetunePolicies($\mathcal{O}$)
7 **return** $\mathcal{O}$

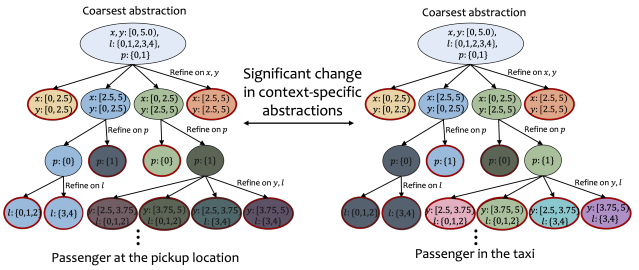

Figure 3: Illustration of two Context-Specific CATs (C-CATs) highlighting different active abstractions (represented by leaves) in the CAT from Fig. 2. The left C-CAT corresponds to $p = 0$, while the right C-CAT corresponds to $p = 1$.

$\Delta_{o_{ij}} = \Delta$, and $\pi_{o_{ij}} = \pi$ (line 5). Finally, we fine-tune the policy for each invented option using CAT+RL (line 6). Options invented in this fashion are used to update the model of abstract options $\mathcal{O}$. We now discuss our method for generating context-specific abstractions (line 3) and identifying option endpoints (ine 4) in detail below.

Generating Context-Specific Abstractions. Recall that the learned CAT captures abstractions for the whole task. We capture context-specific abstractions that depend upon the current state in the form of Context-Specific CATs (C-CATs), which are ablations of the input CAT. To generate C-CATs, we first identify *context-variables* as variables that are relatively more refined among the ones that have a low frequency of change (Hengst et al. (2002)). The degree of refinement for a variable intuitively captures how close the abstraction is to its concrete representation. It is inversely proportional to the measure of reachable concrete values within each value interval of that variable. A high degree of refinement in a variable indicates that it significantly contributes to variation in the Q-function during learning of the CAT. For instance, in the taxi domain, variable p denoting passenger's presence in the taxi is a context-variable since its value persists more and is more refined. The refinement of other frequently changing variables, such as taxi's location, depend on the values of the context-variables. We condition the input CAT on these context-variables to generate C-CATs.

Conditioning the CAT on context-variables highlights abstractions that are salient for different contexts—expressed by different "active" subtrees—within the CAT. More specifically, C-CATs fix values of context-variables in the current state and preserve all abstract states in the CAT that are consistent with these fixed values. All other abstract states that are inconsistent are ignored (shown in dark in Fig. 3). For example, Fig. 3 (left) illustrates the C-CAT for state s_1 where the passenger's location is fixed at the bottom left and the passenger has not yet been picked up, i.e., $s_1(p) = 0$. Similarly, Fig. 3 (right) illustrates the C-CAT for state s_2 where the passenger has been picked up, i.e., $s_2(p) = 1$. We formally define C-CATs as follows.

Definition 3.1 (Context-specific CATs (C-CATs))**.** Given a CAT $\Delta = \langle \mathcal{N}, \mathcal{E} \rangle$ and context-variables $V \subseteq \mathcal{V}$, a *C-CAT* Δ_s for state s is defined as $\langle \mathcal{N}', \mathcal{E}' \rangle$ where $\mathcal{N}' \subseteq \mathcal{N}$ s.t. $\mathcal{N}' = \{\overline{s} | \overline{s} \in \mathcal{N}, v_i \in V, s(v_i) \in \overline{s}(v_i)\}$ and $\mathcal{E}' \subseteq \mathcal{E}$ s.t. $\mathcal{E}' = \{(\overline{s}_1, \overline{s}_2) | (\overline{s}_1, \overline{s}_2) \in \mathcal{E}, \overline{s}_1, \overline{s}_2 \in \mathcal{N}'\}$.

Identifying Option Endpoints. We identify transitions that lead to significant changes in context-specific abstractions to define endpoints of new options. For instance, consider C-CATs in Fig. 2 before and after the passenger is picked up. These C-CATs are significantly different from each other, indicating a significant change in abstraction. To measure difference between abstraction functions represented by two C-CATs generated from the same CAT, we use a context-specific distance function. Intuitively, this distance is computed by traversing from the root node and summing the structural differences between corresponding subtrees of C-CATs. Given a C-CAT Δ_s for state s, let Δ_s^n denote the subtree rooted at node n, and $\text{depth}_{max}(\Delta_s^n)$ denote the maximum depth of that subtree. We drop n from Δ_s^n when $n = s_{init}$. Let n_i denote the ith child of node n in the CAT Δ. We formally define this distance as follows.

Definition 3.2 (Context-specific distance between C-CATs)**.** Given two C-CATs obtained from Δ rooted at node n, $\Delta_{s_1}^n$ and $\Delta_{s_2}^n$, the distance between them is defined as

$$\delta(\Delta_{s_1}^n, \Delta_{s_2}^n) = \begin{cases} \text{depth}_{max}(\Delta_{s_1}^n), & \text{if } n \text{ not in } \Delta_{s_2}^n; \\ \text{depth}_{max}(\Delta_{s_2}^n), & \text{if } n \text{ not in } \Delta_{s_1}^n; \\ \Sigma_i \delta(\Delta_{s_1}^{n_i}, \Delta_{s_2}^{n_i}), & \text{otherwise.} \end{cases}$$

To identify option endpoints using trajectory τ and the context-specific distance, we first identify context-variables from CAT Δ, and generate C-CATs $\Delta_\tau = \langle \Delta_{s_0}, \ldots, \Delta_{s_n} \rangle$. Let δ_{thre} be a distance threshold. Then, for each transition $(s_i, s_{i+1}) \subseteq \tau$, we use abstract states $\phi_\Delta(s_i)$ and $\phi_\Delta(s_{i+1})$ to define option endpoints if $\delta(\Delta_{s_i}, \Delta_{s_{i+1}}) > \delta_{thre}$. The initial and goal abstract states are also included.

Additionally, our approach uses a context-independent distance, also derived from the CAT, to allow decomposing an option into multiple options. For example, navigating to the pickup location can be decomposed into first reaching the larger bottom-left quadrant and then the exact pickup location (Fig. 2). Intuitively, this distance is greater between states that belong to highly distinct (having higher lowest common ancestor (LCA)) and highly refined CAT subtrees. Let depth_{max} be the maximum depth of the CAT, and $\text{depth}(n_1, n_2)$ denote the number of edges between nodes n_1 and n_2. We formally define this distance as follows.

Definition 3.3 (Context-independent distance between abstract states)**.** Given a CAT Δ and LCA of two abstract states $\overline{s}_1$ and $\overline{s}_2$, the distance between them $\sigma_\Delta(\overline{s}_1, \overline{s}_2)$ is defined as the weighted sum of $(\text{depth}_{max} - \text{depth}(\text{root}, LCA) + 1)$ and $(\text{depth}(LCA, \overline{s}_1) + \text{depth}(LCA, \overline{s}_2))/2$.

We decompose options by extending the previously computed sequence of option endpoints as follows. We compute trajectory segments $\tau_{seg} = \langle s_k, \ldots, s_m \rangle \subseteq \tau$ s.t. $\phi_\Delta(s_k)$ and $\phi_\Delta(s_m)$ are consecutive option endpoints. We also compute corresponding abstract trajectory segments $\overline{\tau}_{seg}$ using the CAT. Let $\sigma_{thre} \leq 1.0$ be a distance threshold and σ_{max} be the maximum distance between abstract states in any transition in $\overline{\tau}_{seg}$. Then, for each transition $(\overline{s}_{i\text{-}1}, \overline{s}_i) \subseteq \overline{\tau}_{seg}$, $\overline{s}_i$ is additionally identified as an option endpoint if $\sigma_\Delta(s_{i\text{-}1}, s_i) > \sigma_{thre} \times \sigma_{max}$.

3.3 Planning over Auto-Invented Options

We now describe a novel planning process to compute a plan Π for a new task using the learned model of abstract options and the learned CAT overlayed with abstract transitions, termed as a Plannable-CAT (Alg. 1 line 6). This plannable-CAT is used to guide the search process over the option endpoints, while creating new option signatures to connect them when needed. We apply single-outcome determinization (Yoon, Fern, and Givan 2007) for planning by considering only the most likely effects (here, the termination sets) of options. To compute a plan of options, we first augment the Plannable-CAT with transitions between option endpoints, including lifted transitions between higher levels of abstract states. We then use A^* search over this Plannable-CAT with a cost function that prioritizes lower-level transitions and a heuristic defined by the context-independent distance (Def. 3.3). The idea is to compose abstract transitions at different levels of abstractions. The resulting plan is refined by replacing consecutive higher-level transitions with a new option signature. This helps to bridge gaps between option endpoints. Finally, CHiRP interleaves the execution of the computed plan with learning of option policies for any newly created option signatures to solve the new task.

4 Empirical Evaluation

We evaluated CHiRP[1] on a diverse suite of challenging domains in continual RL setting. Full details about the used domains and hyperparameters are provided in the extended version of our paper (Nayyar and Srivastava 2024).

4.1 Experimental Setup

Domains. For our evaluation, we compiled a suite of test domains for continual RL that are amenable to hierarchichal decomposition and challenging for SOTA methods. We then created versions of these problems that are significantly larger than prior investigations to evaluate whether the presented approaches are able to push the limits of scope and scalability of continual RL. Our investigation focused on *stochastic* versions of the following domains with continuous or hybrid states: (1) **Maze World** (Ramesh, Tomar, and Ravindran 2019): An agent needs to navigate through randomly placed wall obstacles to reach the goal; (2) **Four Rooms World** (Sutton, Precup, and Singh 1999): An agent must move within and between rooms via hallways to reach the goal; (3) **Office World** (Icarte et al. 2018): An agent needs to collect coffee and mail from different rooms and deliver them to an office; (4) **Taxi World** (Dietterich 2000): A taxi needs to pick up a passenger from its pickup location and drop them off at their destination; (5) **Minecraft** (James, Rosman, and Konidaris 2022): An agent must find and mine relevant resources, build intermediate tools, and use them to craft an iron or stone axe.

Baselines. We selected the best-performing contemporary methods that do not require any hand-engineered abstractions or action hierarchies as baselines to match the absence of such requirements in our approach: (1) **Option-Critic** (Bacon, Harb, and Precup 2017) is an end-to-end gradient-based method that learns and transfers option policies and termination conditions; (2) **CAT+RL** (Dadvar, Nayyar, and Srivastava 2023) is a top-to-down abstraction refinement method that dynamically learns state abstractions during RL; and (3) **PPO** (Schulman et al. 2017) is a policy-gradient Deep RL method that progressively learns latent state abstractions through neural network layers.

Hyperparameters. A key strength of CHiRP over baselines is that it requires only five additional hyperparameters beyond standard RL parameters (e.g., decay, learning rate), unlike SOTA DRL methods that need extensive tuning and significant effort in network architecture design. Throughout our experiments, we intuitively set $\delta_{thre} = 0$ and $\sigma_{thre} \sim 1$ to minimize hyperparameter tuning. These values are robust across domains, preventing options from being too small or numerous. We use a limited set of values for k_{cap}, s_{factor}, and e_{max} parameters across different domains to adaptively control the training of an option's policy and CAT. All parameters are set to the same values across a continual stream of tasks. Details on the used hyperparameters for CHiRP and the baselines are provided in the extended version.

Evaluation setting and metrics. We evaluate in a continual RL setting where an agent needs to adapt to changes in initial states, goal states, transition and reward functions. For each domain, 20 tasks are randomly sampled sequentially from a distribution. Each approach is provided a fixed budget of $\mathcal{H}$ timesteps per task before moving on to the next task. Due to stochasticity and lack of transition models, a task is considered solved if the agent achieves the goal $\geq 90\%$ of the time among 100 independent evaluation runs of the learned policy. We report the fraction of tasks solved within the total allocated timesteps for each approach. The reported timesteps include all the interactions with the environment used for learning state abstractions, option endpoints, and option policies. Results are averaged, and standard deviations are computed from 10 independent trials across the entire problem stream.

4.2 Results

We evaluate the presented work across a few key dimensions: sample-efficiency in continual RL setting, and satisfaction of key conceptual desiderata for task decomposition—composability, reusability, and mutual independence.

Q1. *Does CHiRP help improve sample-efficiency over SOTA RL methods in continual RL setting?*

Fig. 4 shows that CHiRP consistently outperforms all baselines. Our results confirm that, while in principle baseline approaches can solve problems without hand-designed abstractions and hierarchies, they require orders of magnitude more data and struggle to solve streams of distinct long-horizon tasks with sparse rewards. We found that CAT+RL delivered the second-best performance, while Option-Critic and PPO consistently underperformed across all domains, failing to solve tasks within the allotted budget. While Option-Critic has the advantage of reusing options, it struggled to learn useful and diverse options. This is at least partly

[1] https://github.com/AAIR-lab/CHiRP

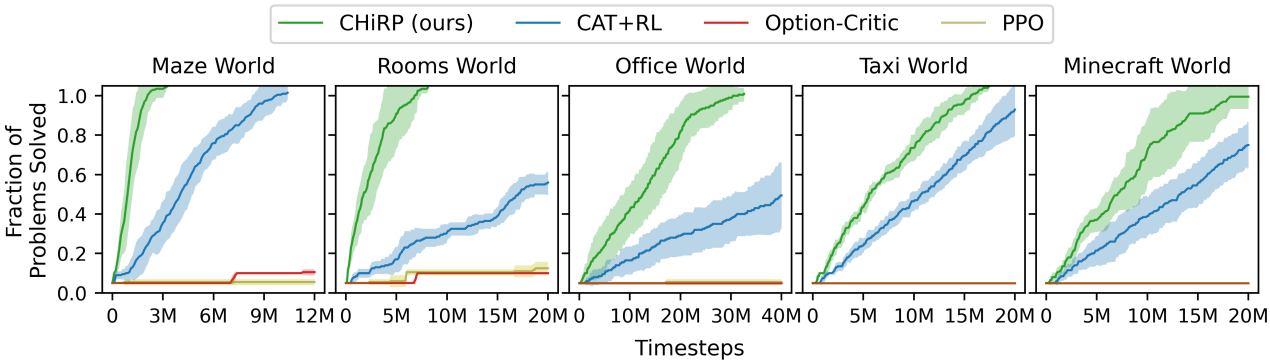

Figure 4: Fraction of tasks solved vs training steps, averaged over 10 independent trials. Each approach was evaluated on a sequence of 20 randomly sampled tasks in a continual learning setting, with a fixed budget of timesteps to solve each task. The timesteps include all environment interactions used for learning both abstractions and policies.

due to lack of mechanisms for modelling initiation sets for options and inability to plan long-term sequences of options, leading to initiation of options in states where they were either ineffective or unnecessary. CAT+RL performed well by learning appropriate state abstractions for each task but did not decompose these tasks into transferable options. PPO struggled to learn likely due to the challenges associated with learning in environments with longer effective horizons and sparse rewards, as shown by Laidlaw, Russell, and Dragan (2023). Gradient-based methods typically rely on dense reward shaping for local gradient updates. In contrast, CHiRP overcomes these limitations by learning options with limited, symbolically represented initiation sets, maintaining option policies at different levels of abstraction, and performing long-term planning with these options. CHiRP benefits from auto-generated state abstractions and goes beyond by inventing reusable options, resulting in more effective generalization and transfer across tasks.

Q2. *Does CHiRP invent mutually independent options? Can the options be composed and reused effectively?*

Options invented by CHiRP have a key advantage: their interpretable symbolic representation, where each option's initiation and termination conditions are defined in terms of specific value ranges of variables. Our analysis revealed that the invented options express distinct, complementary behaviors, with each option primarily affecting different state variables and value ranges. E.g., in the taxi domain, CHiRP invented four options that operate independently: two navigation options that affect different values of taxi location variable and specialize in moving to pickup/drop-off locations, and two passenger interaction options that affect different values of passenger variables and focus on picking up/dropping off the passenger. These options demonstrate mutual independence through minimal overlap in their core affected variables and value ranges in terminations of the options. Their clear symbolic endpoints enable direct chaining of options, making them both composable and reusable.

5 Related Work

Abstraction has been a topic of significant research interest (Karia, Nayyar, and Srivastava 2022; Shah and Srivastava 2024; Shah et al. 2024; Karia et al. 2024). Early research in RL largely focused on hand-designed abstractions (Andre and Russell 2002; Dietterich 2000), with more recent frameworks also using high-level planning models or action hierarchies (Illanes et al. 2020; Kokel et al. 2021). Typically, research on learning abstractions has focused on either state abstraction or action abstraction in isolation (Jonsson and Barto 2000; Wang et al. 2024). A variety of methods have been developed for automatic discovery of subgoals or options, such as identifying bottleneck states through graph-partitioning (Menache, Mannor, and Shimkin 2002; Şimşek and Barto 2007; Machado, Bellemare, and Bowling 2017), clustering (Mannor et al. 2004), and frequency-based (McGovern and Barto 2001; Stolle and Precup 2002) techniques. A large body of work learns hierarchies in which a high-level policy sets subgoals for a lower-level policy to achieve (Vezhnevets et al. 2017; Nachum et al. 2018).

Most prior research in option discovery focuses on control tasks with short horizons, often using dense rewards due to computational intractability (Bacon, Harb, and Precup 2017; Bagaria, Senthil, and Konidaris 2021). Much of this research is limited to single-task settings (Bagaria and Konidaris 2020; Riemer, Liu, and Tesauro 2018). Many recent methods learn a fixed, prespecified number of options and depend on learning a policy over options to use them (Bacon, Harb, and Precup 2017; Machado et al. 2017; Khetarpal et al. 2020; Klissarov and Precup 2021). In contrast, our work tackles a stream of long-horizon, sparse-reward tasks by continually learning generalizable options with abstract representations and planning over them.

6 Conclusion and Future Work

This paper presents a novel approach to continual RL based on autonomously learning and utilizing symbolic abstract options. This work assumes full observability and discrete actions. An interesting future research direction is to extend our approach to settings with continuous parameterized actions. Optimality is another good direction for future work.

References

Abel, D.; Arumugam, D.; Lehnert, L.; and Littman, M. 2018. State abstractions for lifelong reinforcement learning.

In *International Conference on Machine Learning*, 10–19. PMLR.

Andre, D.; and Russell, S. J. 2002. State abstraction for programmable reinforcement learning agents. In *Aaai/iaai*, 119–125.

Andreas, J.; Klein, D.; and Levine, S. 2017. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, 166–175. PMLR.

Bacon, P.-L.; Harb, J.; and Precup, D. 2017. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*.

Bagaria, A.; and Konidaris, G. 2020. Option discovery using deep skill chaining. In *International Conference on Learning Representations*.

Bagaria, A.; Senthil, J. K.; and Konidaris, G. 2021. Skill discovery for exploration and planning using deep skill graphs. In *International Conference on Machine Learning*, 521–531. PMLR.

Barto, A. G.; and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2): 41–77.

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

Dadvar, M.; Nayyar, R. K.; and Srivastava, S. 2023. Conditional abstraction trees for sample-efficient reinforcement learning. In *Uncertainty in Artificial Intelligence*, 485–495. PMLR.

Dietterich, T. 1999. State abstraction in MAXQ hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 12.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of artificial intelligence research*, 13: 227–303.

Hengst, B.; et al. 2002. Discovering hierarchy in reinforcement learning with HEXQ. In *Icml*, volume 19, 243–250. Citeseer.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes*. MIT Press.

Icarte, R. T.; Klassen, T.; Valenzano, R.; and McIlraith, S. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, 2107–2116. PMLR.

Illanes, L.; Yan, X.; Icarte, R. T.; and McIlraith, S. A. 2020. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the international conference on automated planning and scheduling*, volume 30, 540–550.

James, S.; Rosman, B.; and Konidaris, G. 2022. Autonomous learning of object-centric abstractions for high-level planning. In *Proceedings of the The Tenth International Conference on Learning Representations*.

Jin, M.; Ma, Z.; Jin, K.; Zhuo, H. H.; Chen, C.; and Yu, C. 2022. Creativity of ai: Automatic symbolic option discovery for facilitating deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 7042–7050.

Jong, N. K.; and Stone, P. 2005. State Abstraction Discovery from Irrelevant State Variables. In *IJCAI*, volume 8, 752–757.

Jonsson, A.; and Barto, A. 2000. Automated state abstraction for options using the U-tree algorithm. *Advances in neural information processing systems*, 13.

Kaelbling, L. P.; and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, 1470–1477. IEEE.

Karia, R.; Nayyar, R. K.; and Srivastava, S. 2022. Learning generalized policy automata for relational stochastic shortest path problems. *Advances in Neural Information Processing Systems*, 35: 30625–30637.

Karia, R.; Verma, P.; Speranzon, A.; and Srivastava, S. 2024. Epistemic Exploration for Generalizable Planning and Learning in Non-Stationary Settings. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 310–318.

Khetarpal, K.; Klissarov, M.; Chevalier-Boisvert, M.; Bacon, P.-L.; and Precup, D. 2020. Options of interest: Temporal abstraction with interest functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 4444–4451.

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.

Klissarov, M.; and Precup, D. 2021. Flexible option learning. *Advances in Neural Information Processing Systems*, 34: 4632–4646.

Kokel, H.; Manoharan, A.; Natarajan, S.; Ravindran, B.; and Tadepalli, P. 2021. Reprel: Integrating relational planning and reinforcement learning for effective abstraction. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, 533–541.

Laidlaw, C.; Russell, S. J.; and Dragan, A. 2023. Bridging rl theory and practice with the effective horizon. *Advances in Neural Information Processing Systems*, 36: 58953–59007.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a unified theory of state abstraction for MDPs. In *AI&M*.

Liu, B.; Xiao, X.; and Stone, P. 2021. A lifelong learning approach to mobile robot navigation. *IEEE Robotics and Automation Letters*, 6(2): 1090–1096.

Machado, M. C.; Bellemare, M. G.; and Bowling, M. 2017. A laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning*, 2295–2304. PMLR.

Machado, M. C.; Rosenbaum, C.; Guo, X.; Liu, M.; Tesauro, G.; and Campbell, M. 2017. Eigenoption discovery through the deep successor representation. *arXiv preprint arXiv:1710.11089*.

Mannor, S.; Menache, I.; Hoze, A.; and Klein, U. 2004. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, 71.

McGovern, A.; and Barto, A. G. 2001. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. In *Proceedings of the 18th International Conference on Machine Learning, 2001*.

Menache, I.; Mannor, S.; and Shimkin, N. 2002. Q-cut—dynamic discovery of sub-goals in reinforcement learning. In *Machine Learning: ECML 2002: 13th European Conference on Machine Learning Helsinki, Finland, August 19–23, 2002 Proceedings 13*, 295–306. Springer.

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

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Nachum, O.; Gu, S. S.; Lee, H.; and Levine, S. 2018. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31.

Nayyar, R. K.; and Srivastava, S. 2024. Autonomous Option Invention for Continual Hierarchical Reinforcement Learning and Planning (Extended Version). https://aair-lab.github.io/Publications/rkn_aaai25.pdf.

Pateria, S.; Subagdja, B.; Tan, A.-h.; and Quek, C. 2021. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5): 1–35.

Raffin, A.; Hill, A.; Ernestus, M.; Gleave, A.; Kanervisto, A.; and Dormann, N. 2019. Stable baselines3.

Ramesh, R.; Tomar, M.; and Ravindran, B. 2019. Successor options: An option discovery framework for reinforcement learning. *arXiv preprint arXiv:1905.05731*.

Riemer, M.; Liu, M.; and Tesauro, G. 2018. Learning abstract options. *Advances in neural information processing systems*, 31.

Ring, M. B. 1994. *Continual learning in reinforcement environments*. The University of Texas at Austin.

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

Shah, N.; Nagpal, J.; Verma, P.; and Srivastava, S. 2024. From Reals to Logic and Back: Inventing Symbolic Vocabularies, Actions and Models for Planning from Raw Data. *arXiv preprint arXiv:2402.11871*.

Shah, N.; and Srivastava, S. 2024. Hierarchical planning and learning for robots in stochastic settings using zero-shot option invention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 10358–10367.

Şimşek, Ö.; and Barto, A. G. 2007. Betweenness centrality as a basis for forming skills. Technical Report 07-26, University of Massachusetts.

Stolle, M.; and Precup, D. 2002. Learning options in reinforcement learning. In *Abstraction, Reformulation, and Approximation: 5th International Symposium, SARA 2002 Kananaskis, Alberta, Canada August 2–4, 2002 Proceedings 5*, 212–223. Springer.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211.

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*, 3540–3549. PMLR.

Wang, Z.; Wang, C.; Xiao, X.; Zhu, Y.; and Stone, P. 2024. Building minimal and reusable causal state abstractions for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 15778–15786.

Watkins, C. J.; and Dayan, P. 1992. Q-learning. *Machine learning*, 8: 279–292.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A Baseline for Probabilistic Planning. In *ICAPS*, volume 7, 352–359.

7 Appendix

7.1 Selection of Domains and Task Details

We chose continuous state versions of test domains that are well established as challenging for state-of-the-art RL (Andreas, Klein, and Levine 2017; Kokel et al. 2021; Jin et al. 2022). In each domain, we generated a continual stream of 20 randomly sampled tasks, varying in initial states, goal states, transition and reward functions. We evaluated each approach on its ability to solve the entire continual RL problem within a given budget $\mathcal{H}$ for each task. The chosen tasks are particularly difficult due to their long horizons and sparse rewards. We detail the specific differences among the tasks in each randomly generated continual RL problem for the selected domains. These tasks share the same state and action spaces but vary in their objectives as follows.

- In **Maze World** derived from Ramesh, Tomar, and Ravindran (2019), the agent must navigate from a randomly selected starting location to a randomly chosen goal, avoiding randomly placed wall obstacles. It can move in four cardinal directions, with a 0.8 probability of successful movement and a 0.1 probability of slipping to an adjacent cell. The agent receives a reward of 500 upon reaching the goal and -1 elsewhere.
- In **Four Rooms** derived from Sutton, Precup, and Singh (1999), the agent navigates within and between rooms, moving through hallways to reach a randomly chosen goal from a random starting location. The effects of actions due to stochasticity and the reward structure are the same as in Maze World.
- In **Office World** adapted from Icarte et al. (2018), the agent starts at a random location, tasked with collecting coffee and mail from specific locations and delivering them to a randomly chosen desk within an office environment. The agent receives a reward of 500 upon successfully completing a task and 0 elsewhere.
- In **Taxi World** adapted from Dietterich (2000), the taxi starts at a random location, must pick up a passenger from a randomly selected pickup location and drop them off at a randomly chosen destination. The environment features four specific pickup and dropoff locations. The agent is penalized with a -100 reward for illegal pick or dropoff actions, and it receives a reward of 500 for successfully completing the task of delivering the passenger to its destination, with a -1 penalty for other moves.
- In **Minecraft** adapted from James, Rosman, and Konidaris (2022), the agent starts at a random location, must gather wood from randomly selected forest locations to craft a stick, and then mine resources like iron or stone to build either a stone or an iron axe as required. The tasks involve building either a stone or an iron axe, so the agent must identify and collect the necessary resources. Movement incurs a cost of -1, and the agent receives a reward of 500 upon crafting the correct axe.

For training option policies in our approach, we provide the agent an intrinsic sparse reward of 500 upon reaching abstract terminations of options.

7.2 Selection of Baselines

We evaluated existing work to select the best performing baselines on the selected problems, focusing on those that do not rely on hand-engineered abstractions or action hierarchies. Extensive evaluation by (Dadvar, Nayyar, and Srivastava 2023) showed that CAT+RL dominated Qlearning (Watkins and Dayan 1992) and state-of-the-art deep RL approaches like DQN (Mnih et al. 2013), PPO (Schulman et al. 2017), and A2C (Mnih et al. 2016). As a result, we selected CAT+RL as a suitable baseline for comparison. We also compared with PPO (Schulman et al. 2017), a state-of-the-art DRL approach for learning latent representations to solve RL problems. Finally, we selected Option-Critic (Bacon, Harb, and Precup 2017) as it is a state-of-the-art approach for end-to-end learning of transferable options.

7.3 Hyperparameter Details

We implemented our approach in Python. The code for CHiRP[1] is provided with instructions to run. For baselines, we used the open-source code available for Option-Critic[2], CAT+RL[3], and used standard architectures for PPO from Stable-Baselines3[4] by (Raffin et al. 2019).

A key advantage of CHiRP over baselines is that it requires only five additional hyperparameters beyond standard RL parameters (e.g., decay, learning rate) and is robust to most parameter values, as long as they are intuitively set. In contrast, SOTA deep RL methods need extensive tuning, significant effort in network architecture design, $>$15 hyperparameters, and are highly sensitive to hyperparameter values. Unlike Option-Critic, CHiRP does not require the number of options to be specified in advance. For Option-Critic, we parametrize the intra-option policies with Boltzmann distributions and the terminations with sigmoid functions. Tables 1 (Maze World), 2 (Four Rooms), 3 (Taxi World), 4 (Office World), and 3 (Minecraft) show the used hyperparameters for CHiRP and all baselines (the parameters not listed are used default from provided source codes of the baselines).

The parameters used by CHiRP are: δ_{thre} and σ_{thre} are distance thresholds used for option invention, k_{cap} is the bound on the number of abstract states refined in option-specific CATs, s_{factor} is a scaling factor for adaptively adjusting the stepmax for training option policies (stepmax for an option is set to s_{factor} * length of recent successful trajectory), and e_{max} is the maximum episode limit for learning an option's policy before halting training to do replanning. Our approach uses k_{cap} parameter for controlling abstraction refinement for options, similar to its use by CAT+RL for the entire problem. Throughout our experiments, we intuitively set $\delta_{thre} = 0$ and $\sigma_{thre} \sim 1$. These values are robust across domains, preventing options from being too small or numerous. We use a limited set of values for other parameters to minimize hyperparameter tuning. Additionally, all parameters are set to the same values across a continual stream of tasks for a domain.

[1] https://github.com/AAIR-lab/CHiRP

[2] https://github.com/lweitkamp/option-critic-pytorch

[3] https://github.com/AAIR-lab/CAT-RL.git

[4] https://github.com/DLR-RM/stable-baselines3

Hyperparameters	CHiRP	Option-Critic	CAT+RL	PPO
Task budget in timesteps ($\mathcal{H}$)	1.5M	1.5M	1.5M	1.5M
Exploration decay	0.997	0.997	0.997	–
Minimum exploration rate	0.05	0.05	0.05	–
Learning rate (α)	0.05	0.05	0.05	3e-4
Discount factor (γ)	0.99	0.99	0.99	0.99
Episode stepmax	500	500	500	500
Cap on abstraction refinement (k_{cap})	2	–	5	–
Context-specific distance threshold (δ_{thre})	0	–	–	–
Context-independent distance threshold (σ_{thre})	0.95	–	–	–
Factor to adjust stepmax for options (s_{factor})	10	–	–	–
Maximum episodes to halt option training (e_{max})	500	–	–	–
Number of options	–	8	–	–
Temperature	–	0.001	–	–
Termination regularization	–	0.01	–	–
Entropy regularization	–	0.01	–	–
Generalized Advantage Estimator (gae_lambda)	–	–	–	0.95
Steps to run per update (n_steps)	–	–	–	2048
Entropy coefficient (ent_coef)	–	–	–	0.0
Value function coefficient (vf_coef)	–	–	–	0.5
Maximum gradient clipping (max_grad_norm)	–	–	–	0.5

Table 1: Hyperparameters used in Maze World.

Hyperparameters	CHiRP	Option-Critic	CAT+RL	PPO
Task budget in timesteps ($\mathcal{H}$)	2M	2M	2M	2M
Exploration decay	0.998	0.998	0.998	–
Minimum exploration rate	0.05	0.05	0.05	–
Learning rate (α)	0.05	0.05	0.05	3e-4
Discount factor (γ)	0.999	0.999	0.999	0.999
Episode stepmax	800	800	800	800
Cap on abstraction refinement (k_{cap})	2	–	5	–
Context-specific distance threshold (δ_{thre})	0	–	–	–
Context-independent distance threshold (σ_{thre})	0.95	–	–	–
Factor to adjust stepmax for options (s_{factor})	10	–	–	–
Maximum episodes to halt option training (e_{max})	500	–	–	–
Number of options	–	8	–	–
Temperature	–	0.001	–	–
Termination regularization	–	0.01	–	–
Entropy regularization	–	0.01	–	–
Generalized Advantage Estimator (gae_lambda)	–	–	–	0.95
Steps to run per update (n_steps)	–	–	–	2048
Entropy coefficient (ent_coef)	–	–	–	0.0
Value function coefficient (vf_coef)	–	–	–	0.5
Maximum gradient clipping (max_grad_norm)	–	–	–	0.5

Table 2: Hyperparameters used in Four Rooms World.

Hyperparameters	CHiRP	Option-Critic	CAT+RL	PPO
Task budget in timesteps ($\mathcal{H}$)	4M	4M	4M	4M
Exploration decay	0.999	0.999	0.999	–
Minimum exploration rate	0.05	0.05	0.05	–
Learning rate (α)	0.05	0.05	0.05	3e-4
Discount factor (γ)	1	1	1	1
Episode stepmax	1000	1000	1000	1000
Cap on abstraction refinement (k_{cap})	2	–	5	–
Context-specific distance threshold (δ_{thre})	0	–	–	–
Context-independent distance threshold (σ_{thre})	1	–	–	–
Factor to adjust stepmax for options (s_{factor})	4	–	–	–
Maximum episodes to halt option training (e_{max})	200	–	–	–
Number of options	–	8	–	–
Temperature	–	0.001	–	–
Termination regularization	–	0.01	–	–
Entropy regularization	–	0.01	–	–
Generalized Advantage Estimator (gae_lambda)	–	–	–	0.95
Steps to run per update (n_steps)	–	–	–	2048
Entropy coefficient (ent_coef)	–	–	–	0.0
Value function coefficient (vf_coef)	–	–	–	0.5
Maximum gradient clipping (max_grad_norm)	–	–	–	0.5

Table 3: Hyperparameters used in Taxi World.

Hyperparameters	CHiRP	Option-Critic	CAT+RL	PPO
Task budget in timesteps ($\mathcal{H}$)	4M	4M	4M	4M
Exploration decay	0.9991	0.9991	0.9991	–
Minimum exploration rate	0.05	0.05	0.05	–
Learning rate (α)	0.05	0.05	0.05	1e-2
Discount factor (γ)	0.99	0.99	0.99	0.99
Episode stepmax	800	800	800	800
Cap on abstraction refinement (k_{cap})	5	–	5	–
Context-specific distance threshold (δ_{thre})	0	–	–	–
Context-independent distance threshold (σ_{thre})	1	–	–	–
Factor to adjust stepmax for options (s_{factor})	10	–	–	–
Maximum episodes to halt option training (e_{max})	200	–	–	–
Number of options	–	8	–	–
Temperature	–	0.001	–	–
Termination regularization	–	0.01	–	–
Entropy regularization	–	0.01	–	–
Generalized Advantage Estimator (gae_lambda)	–	–	–	0.95
Steps to run per update (n_steps)	–	–	–	2048
Entropy coefficient (ent_coef)	–	–	–	0.0
Value function coefficient (vf_coef)	–	–	–	0.5
Maximum gradient clipping (max_grad_norm)	–	–	–	0.5

Table 4: Hyperparameters used in Office World.

Hyperparameters	CHiRP	Option-Critic	CAT+RL	PPO
Task budget in timesteps ($\mathcal{H}$)	3M	3M	3M	3M
Exploration decay	0.999	0.999	0.999	–
Minimum exploration rate	0.05	0.05	0.05	–
Learning rate (α)	0.05	0.05	0.05	3e-4
Discount factor (γ)	1	1	1	1
Episode stepmax	1000	1000	1000	1000
Cap on abstraction refinement (k_{cap})	2	–	5	–
Context-specific distance threshold (δ_{thre})	0	–	–	–
Context-independent distance threshold (σ_{thre})	1	–	–	–
Factor to adjust stepmax for options (s_{factor})	10	–	–	–
Maximum episodes to halt option training (e_{max})	200	–	–	–
Number of options	–	8	–	–
Temperature	–	0.001	–	–
Termination regularization	–	0.01	–	–
Entropy regularization	–	0.01	–	–
Generalized Advantage Estimator (gae_lambda)	–	–	–	0.95
Steps to run per update (n_steps)	–	–	–	2048
Entropy coefficient (ent_coef)	–	–	–	0.0
Value function coefficient (vf_coef)	–	–	–	0.5
Maximum gradient clipping (max_grad_norm)	–	–	–	0.5

Table 5: Hyperparameters used in Minecraft.