Context-Sensitive Abstractions for RL with Parameterized Actions

Rashmeet Kaur Nayyar[*1], **Naman Shah**[*1,2], **and Siddharth Srivastava**[1]

[1]Arizona State University, Tempe, AZ, USA
[2] Brown Unviersity, Providence, RI, USA
{rmnayyar, shah.naman, siddharths}@asu.edu

Abstract

Real-world sequential decision-making often involves parameterized action spaces that require both, decisions regarding discrete actions and decisions about continuous action parameters governing how an action is executed. Existing approaches exhibit severe limitations in this setting—planning methods demand hand-crafted action models, and standard reinforcement learning (RL) algorithms are designed for either discrete or continuous actions but not both, and the few RL methods that handle parameterized actions typically rely on domain-specific engineering and fail to exploit the latent structure of these spaces. This paper extends the scope of RL algorithms to *long-horizon*, *sparse-reward* settings with parameterized actions by enabling agents to autonomously learn both state and action abstractions online. We introduce algorithms that progressively refine these abstractions during learning, increasing fine-grained detail in the critical regions of the state–action space where greater resolution improves performance. Across several continuous-state, parameterized-action domains, our abstraction-driven approach enables $TD(\lambda)$ to achieve markedly higher sample efficiency than state-of-the-art baselines.

Code — https://github.com/AAIR-lab/PEARL.git

Extended version —
https://aair-lab.github.io/Publications/nss-aaai26.pdf

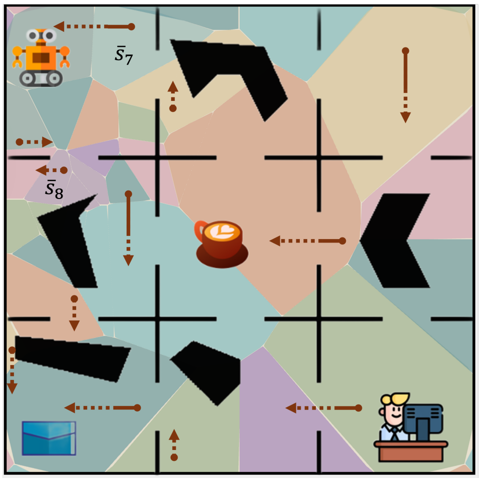


Figure 1: In a continuous version of the office domain, the agent needs to find policies for delivering various items. Polygonal cells illustrate learned state abstractions, and arrows illustrate learned policies with abstract actions. Each arrow represents an interval $[a, b)$ of possible movement values, with a solid line representing a and a dotted segment representing $b - a$ (smaller ranges highlight that higher precision is required).

1 Introduction

Reinforcement learning (RL) has delivered strong results across a diverse range of decision-making tasks, from discrete action settings like Atari games (Mnih et al. 2015) to continuous control scenarios such as robotic manipulation (Schulman et al. 2017). Yet most leading RL approaches (Schulman et al. 2017; Haarnoja et al. 2018; Schrittwieser et al. 2020; Hansen, Su, and Wang 2024) are designed for either discrete or continuous action spaces—not both. Many real-world problems violate this dichotomy. In autonomous driving, for example, the agent must choose among qualitatively distinct actions (accelerate, brake, turn), each endowed with discrete or continuous parameters such as braking force or steering angle. Such actions—known as *parameterized actions*—require choosing not only the action but also determine its (real-valued) parameters before execution.

While recent methods have made progress in addressing parameterized actions (Xiong et al. 2018; Bester, James, and Konidaris 2019; Li et al. 2022), they largely ignore utilizing the underlying structure inherent in parameterized-action spaces. In navigation tasks, for instance, an agent should adjust movement parameters with high precision near obstacles but can act with much coarser control in open areas. Existing approaches also often rely on carefully engineered dense rewards and environment-specific initializations to facilitate learning or benefit from relatively short "effective horizons" to remain tractable (Laidlaw, Russell, and Dragan 2023). A detailed discussion of related work is in Sec. 5.

This paper aims to extend the scope and sample efficiency of RL paradigms to relatively under-studied yet challenging class of problems that feature long horizons, sparse rewards,

[*] These authors contributed equally.

and parameterized actions. We introduce the first known approach called PEARL that automatically discovers structure in parameterized-action problems in the form of conditional abstractions of their state spaces and action spaces. As an illustration, Fig. 1 shows flexible abstraction of the state space and how the policy may require a different extent of action abstraction in different states in the OfficeWorld domain: in the tightly constrained region $\overline{s}_8$, navigation demands high-precision action parameters, whereas the more open space of $\overline{s}_7$ tolerates far coarser abstraction. This contrast highlights why abstractions must capture this variation in the required precision of action parameters across different regions of the state space.

Given an input problem in the RL setting where a state is expressed using discrete and continuous state variables and an action is expressed using continuous or discrete parameters, PEARL learns context-sensitive abstractions while performing $TD(\lambda)$. It uses a combination of dispersion in TD-error and value-function signals to learn which abstract states and action parameters require finer resolution during learning. Our approach builds upon our recent work on conditional state abstractions (Dadvar, Nayyar, and Srivastava 2023), and introduces new algorithms for learning more general forms of state abstractions alongside abstractions of action parameters.

Our main contributions are: (1) A unifying formal framework for context-sensitive abstractions of continuous state spaces and parameterized actions with continuous arguments; (2) an approach for learning flexible refinements of abstractions; (3) algorithms for learning such state and action abstractions on the fly, during RL, and thereby exploiting latent structural properties of problem instances for efficient learning without any hand-crafting of abstractions; (4) an evaluation of this approach as applied to $TD(\lambda)$, showing that using this abstraction paradigm with foundational RL paradigms improves their performance beyond state-of-the-art algorithms.

2 Preliminaries

We use the framework of episodic factored goal-oriented Markov decision process (MDP) with parameterized actions (Bertsekas et al. 2011; Hausknecht and Stone 2016; Deng, Devic, and Juba 2022). An MDP $\mathcal{M}$ is defined as $\langle \mathcal{V}, \mathcal{S}, \mathcal{A}, T, R, \gamma, h, s_0, G \rangle$, where $\mathcal{V}$ is a set of state variables and the domain of each variable $v \in \mathcal{V}$ is a bounded interval $\mathcal{D}_{v_i} = \left[\mathcal{D}^{min}_{v_i}, \mathcal{D}^{max}_{v_i}\right] \subseteq \mathbb{R}$; $\mathcal{S}$ denotes the set of factored states defined by $\mathcal{V}$, where a state $s \in \mathcal{S}$ is an assignment of values to all variables in $\mathcal{V}$: $s = \{v_i = x_k | v_i \in \mathcal{V} \wedge x_k \in \mathcal{D}_{v_i}\}$. We use $s(v_i)$ to denote the value of variable v_i in state s.

The action set $\mathcal{A}$ consists of a finite number of stochastic parameterized actions. Each action $a \in \mathcal{A}$ is a parameterized function $a_l(a_p)$, where a_l is the action label and $a_p = \langle x_1, \ldots, x_k \rangle$ is an ordered set of k continuous parameters where each parameter x_i has a bounded and ordered domain $\mathcal{D}_{x_i} \subseteq \mathbb{R}$. The complete parameter space is defined as $\mathcal{P}_a = \bigtimes_{i=1}^{k} \mathcal{D}_{x_i}$. A grounded action $\tilde{a_i}$ assigns values to these parameters from their respective domains. The set of all possible grounded actions is denoted $\tilde{A}$, and may be infinite given continuous parameters.

The transition function $T : \mathcal{S} \times \tilde{\mathcal{A}} \rightarrow \mu\mathcal{S}$ defines a distribution over next states, given a state and a grounded action. The reward function $R : \mathcal{S} \times \tilde{\mathcal{A}} \rightarrow \mathbb{R}$ assigns scalar rewards to state-action pairs. The discount factor $\gamma \in [0, 1]$ determines the weights of future rewards, and h is the episode horizon. s_0 is the initial state and G is the set of goal states.

The objective is to learn a policy $\pi_{\mathcal{M}} : \mathcal{S} \rightarrow \tilde{\mathcal{A}}$ that when executed from the initial state s_0, reaches a goal state in $s_g \in G$ while maximizing the expected cumulative discounted reward $\mathbb{E}_\pi[\sum_{t=0}^{t=h} \gamma^t r_t]$. We use the RL setting, where both T and R are unknown (Sutton and Barto 1998).

The state-value function $V^\pi(s)$ under a policy π denotes the expected return starting from state s and following π:

$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{h} \gamma^t r_t \mid s_0 = s\right]$$

The action-value function $Q^\pi(s, \tilde{a})$ gives the expected return starting from state s, executing action $\tilde{a}$, and thereafter following π:

$$Q^\pi(s, \tilde{a}) = \mathbb{E}_\pi\left[\sum_{t=0}^{h} \gamma^t r_t \mid s_0 = s, \tilde{a}_0 = \tilde{a}\right]$$

TD(λ) We use TD(λ) (Sutton 1988) for learning the policy π. It combines one-step TD and Monte Carlo methods by weighting updates across multiple future time steps, controlled by the trace-decay parameter $\lambda \in [0, 1]$.

Abstraction Abstraction has been recognized as a key mechanism for achieving scalability in long horizon, sparse reward settings (Li, Walsh, and Littman 2006; Shah and Srivastava 2024; Wang et al. 2024). A state abstraction is a mapping $\alpha : \mathcal{S} \rightarrow \overline{\mathcal{S}}$ that assigns each concrete state $s \in \mathcal{S}$ to an abstract state $\overline{s} \in \overline{\mathcal{S}}$, where $\overline{\mathcal{S}}$ is a partitioning of the original state space $\mathcal{S}$. In this work, we define an analogous notion of abstraction for an action, defined as a partitioning of the action-parameter space (formalized in Sec. 3.1).

We now describe our approach for efficiently learning a policy in settings with parameterized actions by automatically learning context-sensitive state and action abstractions.

3 Our Approach

The central contribution of this paper is a novel abstraction paradigm for jointly representing and learning state and action abstractions. These abstractions exploit the structure of the environment in order to efficiently learn and represent policies for problems with parameterized actions.

Running Example Consider an AI agent in an Office environment (Fig. 1) that must collect and deliver a coffee and a mail between rooms and offices. The state variables include the agent's (x, y) position with $x, y \in [0.0, 5.0)$, and two binary variables: $c \in \{0, 1\}$ and $m \in \{0, 1\}$ indicating whether it is carrying coffee or mail. The agent has four actions to move in the cardinal directions, i.e.,

$\mathcal{A} = \{up(d), down(d), left(d), right(d)\}$, each with one continuous parameter $d \in [0, 0.5)$ that determines the movement distance. Actions may result in stochastic displacements along orthogonal directions, and the agent picks or drops items automatically at designated locations. This setting extends the OfficeWorld environment (Icarte et al. 2022) by incorporating parameterized actions.

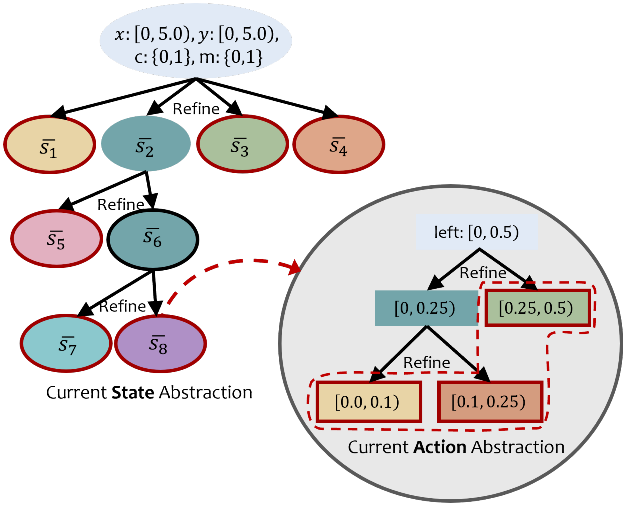


Figure 2: Illustration of a SPA-CAT for Office World.

This work builds upon our prior work (Dadvar, Nayyar, and Srivastava 2023) which learned state abstractions with strictly uniform refinements. It did not address the problem of parameterized actions and offered no mechanism for more flexible refinements of abstractions. The abstraction learned in this work has the following desirable properties: (i) The abstractions are flexible—the abstract state boundaries are not constrained to be orthogonal or axis-aligned. This flexibility allows the learned abstractions to better adapt to the geometry and dynamics of the environment—for example, by placing boundaries where agent behavior changes, following the contours of obstacles. Fig. 1 illustrates an example of such a state abstraction for Office World, where each colored region represents a distinct abstract state. (ii) Moreover, each abstract state has a conjoined action parameter tree for each action (shown in Fig. 2), allowing varying levels of precision in different abstract states. E.g., in open areas—such as the centers of rooms (e.g., abstract state $\bar{s}_7$)—the agent can move freely without requiring high precision in selecting movement distances (e.g., abstract action *left*([0.25,0.5))). In contrast, in more constrained areas—such as corridors, near obstacles, or narrow passages (e.g., abstract state $\bar{s}_8$)—precise control over movement is crucial (e.g., abstract action *left*([0.0,0.1))). Fig. 2 shows the unified state-action abstraction tree, where the leaves represent these abstract states and abstract actions. The shown abstractions capture the required higher precision for selecting action parameters. We hypothesize that automatically identifying such meaningful abstract states and corresponding action parameter trees can significantly improve sample efficiency in policy learning.

We now define our unified framework for jointly representing both state and action abstractions.

3.1 State and Action Abstractions

This section formalizes our representations for state and action abstractions, beginning with action parameter abstractions, followed by an integrated representation for state and action abstractions. We use *action parameter trees* (APTs) to formalize the intuitive example of action parameter abstractions discussed above. Each node in an APT represents a susbset of the parameter space of an action, and its children nodes together represent a partition of that subset. Formally, given a parameterized action $a \in \mathcal{A}$ with a complete parameter space $\mathcal{P}_a$, we define a corresponding APT as follows:

Definition 3.1 (Action Parameter Tree (APT))**.** An APT τ is a directed hierarchical structure defined as a tuple $\langle \mathcal{N}, \mathcal{E}, N_0, \ell \rangle$ where $\mathcal{N}$ is a set of nodes, $\mathcal{E}$ is a set of edges such that each $(u, v) \in \mathcal{E}$ represents a directed edge from node u to node j. N_0 is the root node. $\ell : \mathcal{N} \rightarrow 2^{\mathcal{P}}$ defines a labeling function that maps each node $n_i \in \mathcal{N}$ to a subset of the parameter space such that $\ell(N_0) = \mathcal{P}$, the complete set of parameter values. The set of all children nodes $\{n_j\}_{\{j=1,\ldots,k\}}$ of node n_i represent a partition of $\ell(n_i)$, i.e, $\cup_{j=1,\ldots,k}\ \ell(n_j) = \ell(n_i)$ with labels of children nodes representing mutually exclusive sets.

Given an APT τ_a for a parameterized action $a \in \mathcal{A}$, we define $\mathcal{L}_\tau \subseteq \mathcal{N}_\tau$ as the set of leaf nodes or the "fringe" of τ_a. The tree structure is learned autonomously so that at any stage of learning, the fringe of an action's APT represents the current abstraction of its parameter space. In this way, the fringe of τ_a can be used to define a set of abstractly grounded versions of a, where each version picks its parameters from one of the leaves of $\mathcal{L}_\tau$.

Formally, the set of abstract parameter sets defined by an APT τ for an action labeled a is defined as $\bar{\mathcal{A}}_\tau = \{\ell(n) | n \in \mathcal{L}_\tau\}$. Given a concrete action $a(q)$ and an APT τ for a, we use $\bar{q}_\tau$ to denote the unique element of $\bar{\mathcal{A}}_\tau$ that includes q. Thus, $a(\bar{q}_\tau)$ denotes the abstraction of $a(q)$ under τ. For brevity, we will use the form $\bar{a}$ to denote an abstract version of a, and $\tilde{a}$ to denote its concrete grounded version (henceforth referred to as a "concrete action") with real-valued parameters. During RL, we execute an abstract action $\bar{a} = a(\bar{q}_\tau)$ by sampling its parameters q uniformly from $\bar{q}_\tau$ to obtain a concrete executable action $\tilde{a} = a(q)$.

We define unified state and action abstractions using state and parameterized-action conditional abstraction trees (SPA-CATs). Intuitively, each node of a SPA-CAT defines a subset of the state space and is associated with its own APTs for each action. The structure of the state abstraction part of the tree is congruent with our notion of APTs but applied to the state space. Formally,

Definition 3.2 (State and Parameterized Action Conditional Abstraction Tree (SPA-CAT))**.** A SPA-CAT Δ is a directed hierarchical structure defined as a tuple $\langle \mathcal{N}, \mathcal{E}, N_0, \ell_s, \ell_a, \rangle$, where $\mathcal{N}$ is a set of nodes, $\mathcal{E}$ is a set of directed edges. Each edge $(u, v) \in \mathcal{E}$ defines a directed edge between nodes u to v. $N_0 \in \mathcal{N}$ defines a root node without a parent node. $\ell_s : \mathcal{N} \rightarrow 2^{\mathcal{S}}$ defines a labeling function that maps each node $n \in \mathcal{N}$ to a subset of the state space $\mathcal{S}_n \subseteq \mathcal{S}$ such that $\ell_s(N_0) = \mathcal{S}$. The set of all children nodes $\{n_j\}_{\{j=1,\dots,k\}}$ of a node n_i repesent a partition of $\ell(n_i)$, i.e., $\cup_{j=1\dots,k}\, \ell_s(n_j) = \ell_s(n_i)$, with labels of children nodes representing mutually exclusive sets. $\ell_a : \mathcal{N} \times \mathcal{A} \rightarrow \Theta$ maps node $n_i \in \mathcal{N}$ and a parameterized action $a_j \in \mathcal{A}$ to an APT τ_{a_j} in the set of all possible APTs, Θ.

SPA-CATs define state and action abstractions as follows. The set of leaf nodes (or the "fringe") of a SPA-CAT Δ, denoted as $\mathcal{L}_\Delta \subseteq \mathcal{N}_\Delta$, define an abstract state space: $\bar{\mathcal{S}}_\Delta = \{\ell_s(n) | n \in \mathcal{L}_\Delta\}$. Let $n_\Delta(s)$ denote the unique fringe node of Δ that represents s. The *abstraction of a concrete state* s under Δ, $\bar{s}_\Delta$, is defined as the set represented by the unique fringe element that includes s: $\bar{s}_\Delta = \ell_s(n_\Delta(s))$. Further, each node n in the fringe is associated with an APT $\ell(n, a)$ for each $a \in \mathcal{A}$. This allows us to define the abstraction of a grounded action $a(q)$ relative to a concrete state s and a SPA-CAT Δ, $\bar{a}_{s,\Delta}$, as $a(\bar{q}_\tau)$, where $\tau = \ell(n_\Delta(s), a)$. We omit subscripts when clear from context. In this representation, each abstract state defines its own APTs. This allows the agent to tune the level of precision in each action's abstraction as a function of the current state. This is particularly conducive for compact expressions of $Q(s, a)$ functions in RL. We now discuss our approach for automatically learning SPA-CATs from scratch during RL.

3.2 Learning Abstraction Trees

Throughout this work, we use abstraction trees introduced above to express Q functions. In particular, we express and maintain an abstract Q function as a mapping from the abstract states and actions defined by a SPA-CAT (Sec. 3.1) to $\mathbb{R}$. This allows generalization over unseen state-action pairs using the Q values for their abstractions. This section describes our approach for learning SPA-CATs using state-action trajectories collected using any sequential decision making algorithm; our overall algorithm integrating the decision-making process, data collection, and the invocation of SPA-CAT learning phases is discussed in the next section.

SPA-CATs are learned autonomously through a process of hierarchical refinement. The SPA-CAT Δ is initialized with the universal abstraction where Δ has a single node corresponding to the entire state space, and each action's APT associated with this node has a single node capturing that action's entire parameter space. The refinement process creates children nodes for nodes at the fringes of the SPA-CAT and at the fringes of the APTs associated with SPA-CAT nodes. These refinements increase the granularity of abstraction in regions of the state and action spaces where finer distinctions are necessary for high performance decision-making.

Suppose the RL agent encounters a set of execution traces of the form $\mathcal{D} = \{\langle s_0, a_0, r_0, \dots s_n, a_n, r_n\rangle\}$ where s_i is a concrete state, a_i is an action executed in s_i, and r_i is the incurred reward for the transition. These traces are abstracted using the current version of Δ to produce $\bar{\mathcal{D}} = \{\langle \overline{s_0}_\Delta, \overline{a_0}_{s_0,\Delta}, \bar{r}_0, \dots, \overline{s_n}_\Delta, \overline{a_m}_{s_m,\Delta}, \bar{r}_m\rangle\}$. The abstract sequence is constructed to avoid consecutive duplicate abstract states: trajectory subsequences $\langle s_i, a_i, r_i, \dots s_{i+k}, a_{i+k}, r_{i+k}\rangle$ whose state-action segments are abstracted to the same pair are represented only once as $\langle \overline{s_i}_\Delta, \overline{a_i}_{s_i,\Delta}, \bar{r}_i\rangle$, where $\bar{r}_i$ is the total cumulative discounted reward for the original subsequence.

The learning process aims to create agglomerative abstractions where regions of the state and action space that portend similar futures are grouped together. This indicates that dispersions in the value function estimates of abstract states could be used to identify areas where heterogeneous states are incorrectly being combined into an abstraction. However, during early stages of learning, the agent's policy can vary significantly, and the paucity of data makes value-function estimates extremely unreliable as indicators of similarity in future courses of action. To balance these considerations, we define a novel hybrid formulation of heterogeneity to identify elements of the current state-action abstraction that need to be refined.

Given abstract traces $\bar{\mathcal{D}}$, the TD error $\delta(\bar{s}_i, \bar{a}_i)$ for each subsequent abstract state and action is defined as follows:

$$\delta(\bar{s}_i, \bar{a}_i) = (\bar{r}_i + \gamma \max_{\bar{a}} \overline{Q}(\bar{s}_{i+1}, \bar{a})) - \overline{Q}(\bar{s}_i, \bar{a}_i) \tag{1}$$

It is well-known that value-function is inaccurate at start of Q-learning, thus unsuitable for early refinement, whereas, TD-error better represents similar futures during early learning (e.g., (Kearns and Singh 1998)). Thus, in early stages of learning, we rely on the value of the temporal difference (TD) error to provide a stronger signal of behavioral inconsistency: if $\delta(\bar{s}, \bar{a})$ values show a high standard deviation (SD) for an abstract state-action pair in $\bar{\mathcal{D}}$, then this abstract pair may be representing heterogeneous regions where the Q function is changing at significantly different rates. On the other hand, as learning progresses and the policy stabilizes, value function estimates become more reliable. At this point the variability of value-function estimates across concrete states in an abstract state are a better indicator of heterogeneity in the abstraction. Therefore, we blend TD error and value-function dispersion metrics. Since it is infeasible to maintain a tabular representation (e.g., a Q-table) over all continuous concrete states and actions, we compute an estimate of the concrete-state value function as follows:

$$\hat{V}(s_i) = r_i + \gamma \max_{\bar{a}} \overline{Q}(\bar{s}_{i+1}, \bar{a}) - Q(s_i, \bar{a}) \tag{2}$$

This allows us to estimate the value function for a state using a learned Q-value function for abstract states and abstract actions. We capture the dispersion of $\hat{V}$ estimates across all n concrete states s_i present in the dataset $\mathcal{D}$.

We combine the standard deviation over TD errors and over V function estimates into a novel heterogeneity estimate for state-action pairs in $\bar{\mathcal{D}}$:

$$\begin{aligned} H(\bar{s}_i, \bar{a}_i) = \beta \cdot SD_{\bar{\mathcal{D}}}\left[\delta(\bar{s}_i, \bar{a}_i)\right] + \\ (1 - \beta) \cdot SD_{\mathcal{D}}\left[\hat{V}(s_i)\right]_{s_i \in \bar{s}_i} \end{aligned} \tag{3}$$

Here, the standard deviation is computed over all occurrences of the pair $\bar{s}_i, \bar{a}_i$ in $\bar{\mathcal{D}}$. A scheduling mechanism is used to gradually shift emphasis from TD error dispersions to value function dispersions. This is achieved with a weighting parameter β, initialized at 1.0 and annealed over

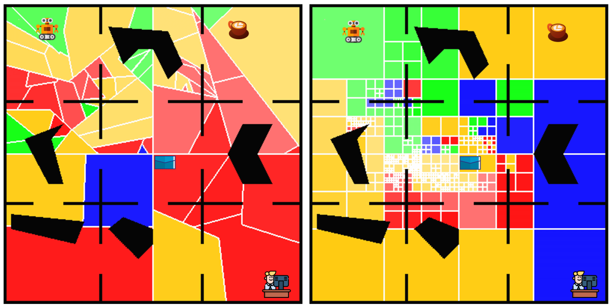


Figure 3: Learned state abstractions using flexible (left) and uniform (right) refinement strategies. The agent is at the top left; it must deliver both coffee and mail to the bottom right. Black lines and regions indicate obstacles. Colors represent actions (yellow: right, green: down, red: up, blue: left).

time. High heterogeneity values prompt the refinement of the abstract state into finer abstractions.

We rank each abstract state-action pair using the computed heterogeneity H and select top-k abstract states and abstract actions to refine. We use $H(\overline{s}) = \max_{\overline{a}} H(\overline{s}, \overline{a})$ to select abstract states for refinement and use $H(\overline{s}, \overline{a})$ for selecting abstract actions for refinements.

Multiple paradigms can be used for refining the abstract state-action regions that feature a high heterogeneity under this formulation. We consider two paradigms: uniform refinement as proposed in prior work (Dadvar, Nayyar, and Srivastava 2023), and a novel flexible refinement that uses statistical learning. Both state abstractions (nodes for SPA-CATs) and action abstractions (nodes for Action Trees) can be refined using these methods. However, for brevity, we describe them in the context of refining state abstractions.

Uniform refinement Given an abstract state $\overline{s}$ selected for refinement, uniform partitioning bisects the interval corresponding to each (variable) independently, resulting in an orthogonal binary tree decomposition of the state space (see Fig. 3 (right)). While straightforward, such abstractions are best suited for domains where the Q-function can be factorized into functions over individual state variables and they require extensive refinements to express regions that feature homogeneous value function estimates, but do not constitute hypercubes.

Algorithm 1: PEARL

```
Input: MDP $\mathcal{M} = \langle \mathcal{V}, \mathcal{S}, \mathcal{A}, T, R, \gamma, h \rangle$
Output: Policy $\pi$ for MDP $\mathcal{M}$ and SPA-CAT $\Delta$
Initialize SPA-CAT $\Delta$ and Qtable $\overline{Q}$
Initialize buffers $D_{\overline{s},\overline{a}}$ and $D_{s,\overline{a}}$
for $episode = 1 : n_{epi}$ do
    // Learning phase
    $s \leftarrow$ reset()
    for $step = 1 : h$ do
        $\overline{a} \leftarrow \pi(\overline{Q}, \overline{s})$
        $\overline{s}', \overline{r}, \{s_i, \overline{a}_i, r_i, \dots, s_k\} \leftarrow$ execute$(s, \overline{a})$
        $\mathcal{D}_{s,\overline{a}}$.add$(\{s_i, \overline{a}_i, r_i, \dots, s_k\})$
        $\mathcal{D}_{\overline{s},\overline{a}}$.add$(\{\overline{s}, \overline{a}, \overline{r}, \overline{s}'\})$
        $\overline{Q} \leftarrow$ updateQvalue$(\overline{s}, \overline{a}, \overline{r}, \overline{s}')$
        $V \leftarrow$ updateValue$(s, \overline{a}, r, s', \overline{s}')$
    // Refinement phase
    if $episode$ mod $n_{refine} = 0$ then
        $\mathcal{D}_{\overline{s},\overline{a}} \leftarrow$ computeHeterogeneity$(\overline{Q}, \mathcal{D}_{\overline{s},\overline{a}})$
        $\bar{\mathcal{S}}_{ref}, \bar{\mathcal{A}}_{ref} \leftarrow$ findImprecise$(\mathcal{D}_{\overline{s},\overline{a}})$
        if $refinement == flexible$ then
            $\mathcal{D}_{s,\overline{a}} \leftarrow$
              estimateSimilarity$(V, \mathcal{D}_{s,\overline{a}}, \bar{\mathcal{S}}_{ref}, \bar{\mathcal{A}}_{ref})$
            $\mathcal{C} \leftarrow$ cluster$(\bar{\mathcal{S}}_{ref}, \mathcal{D}_{s,\overline{a}})$
            $\Delta \leftarrow$ refine$(\mathcal{C}, \bar{\mathcal{S}}_{ref}, \bar{\mathcal{A}}_{ref})$
        else
            $\Delta \leftarrow$ refine$(\bar{\mathcal{S}}_{ref}, \bar{\mathcal{A}}_{ref})$
        Reinitialize $\mathcal{D}_{\overline{s},\overline{a}}$ and $\mathcal{D}_{s,\overline{a}}$
return $\pi$, $\Delta$
```

Flexible refinement We introduce a novel learning-based approach for constructing flexible refinements. Given an abstract state $\overline{s}$ selected for refinement and the associated set of execution traces, we partition $\overline{s}$ into at most $\mathcal{K}$ finer abstract states by clustering the concrete states contained within $\overline{s}$. Specifically, we apply Agglomerative Clustering (Murtagh and Contreras 2012) from scikit-learn (Pedregosa et al. 2011) with an adaptive distance threshold: starting from 0.1, we incrementally increase the threshold by 0.001 until the number of clusters is below a specified maximum. This prevents over-fragmentation while ensuring meaningful behavioral distinctions are captured. We use the following similarity criterion to form coherent partitions that reflect underlying behavioral distinctions:

$$
\begin{aligned}
J(s) &= \beta \cdot \hat{\delta}(s) + (1-\beta) \cdot \hat{V}(s) \\
\hat{\delta}(s_i) &= r(s_i, \overline{a}) + \gamma \max_{\overline{a}'} \overline{Q}(\overline{s}_{i+1}, \overline{a}') - Q(s_i, \overline{a}_i), \qquad (4) \\
\text{where } \overline{a} &= \arg\max_{\overline{a}} H(\overline{s}, \overline{a})
\end{aligned}
$$

Here, $\hat{\delta}$ and $\hat{V}$ are estimated TD-errors and state values for a concrete state, and $\overline{a}$ is the abstract action with high heterogeneity for an abstract state $\overline{s}$. We use a schedule similar to the heterogeneity estimate shift to prioritize using TD-errors estimates earlier in the learning and state-values later in the learning using an annealed parameter β.

Once the partitions are identified, we train an SVM classifier to learn decision boundaries between them and define abstract states, with each partition corresponding to a new abstract state. We use balanced class weights and select the regularization parameter through cross-validation based on the smallest class size, evaluating both RBF and linear kernels. This yields refined abstract states that more effectively capture variations in decision-relevant signals like TD error and value estimates, enabling more expressive abstractions.

We now discuss our algorithm for autonomously learning a SPA-CAT and a policy for a given problem.

3.3 PEARL Algorithm

Alg. 1 (Parameterized Extended state/action Abstractions for Reinforcement Learning, PEARL), presents the overall process for integrating the learning of SPA-CATs with TD(λ). It begins with an initial, coarse SPA-CAT with a single node N_0 and one APT for each action $a \in \mathcal{A}$ with single nodes each (line 1). PEARL allows the agent to execute in the environment, while collecting its trajectories and incrementally refining the SPA-CAT as discussed in the preceding section. This allows PEARL to simultaneously learn a SPA-CAT and a solution policy for the input MDP $\mathcal{M}$. It has two main phases: (a) a learning phase (lines 4-11), where a policy is trained while keeping the SPA-CAT Δ fixed, and (b) a refinement phase (lines 12-20), where the abstraction is improved by refining the current SPA-CAT. We now disucss these two phases in detail.

Learning phase In this phase, the agent learns an abstract policy $\pi : \overline{\mathcal{S}} \to \overline{\mathcal{A}}$ over the current SPA-CAT structure using tabular TD-λ (Sutton 1988) for n_{refine} episodes (lines 4–11). During each episode, the agent follows the abstract policy by executing the corresponding abstract action in the current abstract state, continuing until it reaches a new abstract state or the episode terminates (lines 7–8).

Traces obtained during execution are used to update Q-values and TD errors over abstract states and actions using standard TD(λ) updates (lines 10–11), enabling policy improvement in the abstract state space.

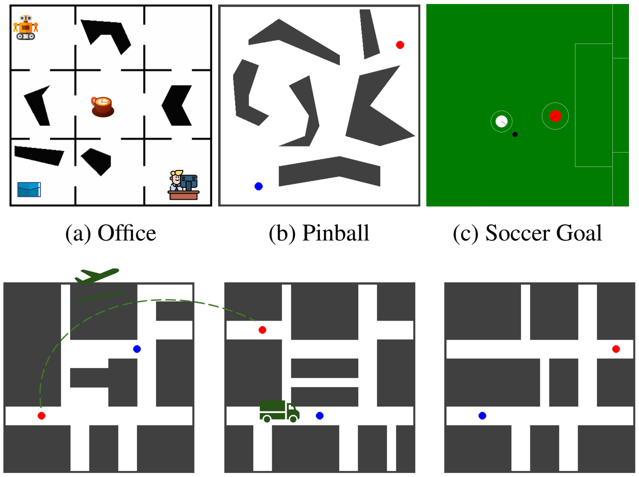
Refinement phase After every n_{refine} episodes, PEARL enters the refinement phase to update the SPA-CAT (lines 12-21) via heterogeneity and similarity measures computed using the methods presented in Sec. 3.2. The updated SPA-CAT is then used to continue the learning phase.

We now discuss thorough empirical evaluation of our approach in a variety of settings with parameterized actions.

4 Empirical Results

We implemented PEARL along with the annealed heterogeneity estimation and abstraction refinement paradigm presented above. This implementation uses a flexible refinement strategy for refining SPA-CATs and a uniform refinement strategy for refining APTs. We evaluate PEARL along three key dimensions: (1) improvements in sample-efficiency, (2) the quality of the learned policies, and (3) the size of the abstractions generated. Our evaluation is conducted across four challenging SOTA RL domains with stochastic and unknown action models, continuous states, parameterized actions, and sparse rewards (a positive reward only upon reaching the goal). Combined with long-horizons, these tasks represent significant challenges for RL.

Test environments We evaluate on domains well-established as challenging (illustrated in Fig. 4): (i) OfficeWorld (Icarte et al. 2022; Corazza et al. 2024) (ii) Pinball (Roice et al. 2024; Rodriguez-Sanchez and Konidaris 2024), (iii) Multi-city transport (Ma et al. 2021; Oswald et al. 2024), and (iv) Robot Soccer Goal (Bester, James, and Konidaris 2019). Among these, former three are especially challenging due to longer effective planning horizons.



(a) Office

(b) Pinball

(c) Soccer Goal

(d) Multi-City Transport

Figure 4: (a) Office World: The robot needs to pickup coffee and mail and deliver to the office. (b) Pinball: A small, dynamic ball needs to be manouvered into a red hole, avoiding collisions with irregularly shaped obstacles. (c) Soccer Goal: The white agent needs to kick the small black ball past the red keeper. (d) Multi-City Transport: The agent needs to collect a package from a designated location (marked by blue) in a city and deliver to a target airport (marked by red) in a different city. Cities are connected only via airports.

Baseline selection Standard RL approaches—tabular RL (Sutton 1988; Watkins et al. 1989), deep RL (Mnih et al. 2015; Lillicrap et al. 2015; Schulman et al. 2017; Haarnoja et al. 2018), hierarchical RL (Nachum et al. 2018; Levy et al. 2019)—are not designed to handle parameterized actions, making them unsuitable as baselines. We therefore compare PEARL against two baselines that support parameterized actions: (i) MP-DQN (Bester, James, and Konidaris 2019), which extends P-DQN (Xiong et al. 2018) by combining DQN and DDPG while addressing P-DQN's overparameterization problem through multi-pass processing, and (ii) HyAR (Li et al. 2022), which learns latent space of hybrid action space and models dependencies between discrete action and continuous parameter using an embedding table and a conditional Variational Auto-Encoder (VAE). To evaluate and compare learning performance fairly without manually biasing learning with head-starts towards favorable solutions, we used the original source code for these baselines while removing their hand-crafted, environment-specific weight initializations. We replaced them with zero or randomized initializations, whichever yielded better performance. This provides a consistent, unbiased evaluation of each method's true learning capability.

Metrics and hyperparameters We evaluate all agents using two key metrics: (i) cumulative average return during training, and (ii) the success rate of the learned greedy policy. We evaluate two variants of PEARL—PEARL-flexible and PEARL-uniform—which differ in their approach to learning abstractions. Reported performance of PEARL variants include all episodic interactions used for learning

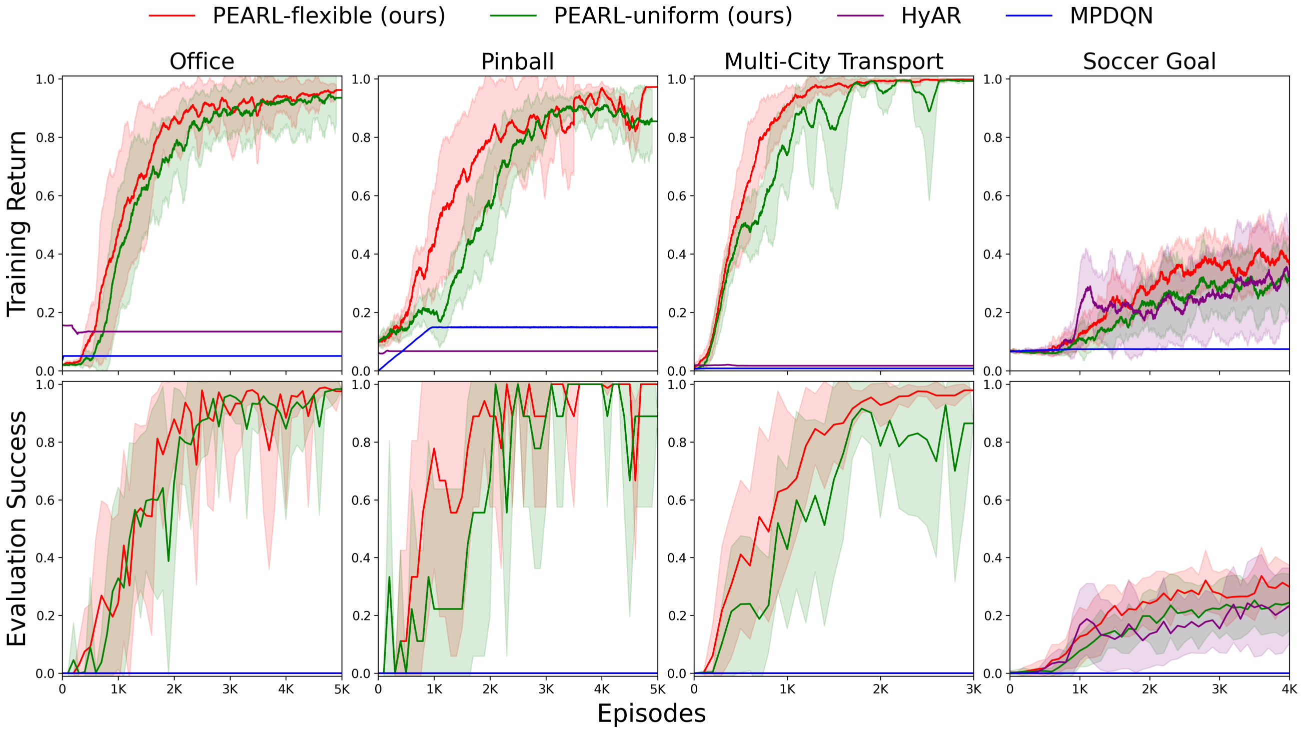


Figure 5: Comparison of PEARL-flexible and PEARL-uniform with MP-DQN and HyAR in four domains: Office World, Pinball, Multi-City Transport, and Soccer Goal with mean and standard deviation across 50 independent trials.

state and action abstractions. The results are averaged over 50 independent runs, with both mean and standard deviation reported. Full hyperparameter details for all methods are provided in the extended version.

4.1 Analysis of the results

Sample efficiency and performance Fig. 5 shows the performance of all methods, with training episodes on the x-axis and two rows of metrics on the y-axis: cumulative return (during training) and success probability (during evaluation). Despite learning abstractions from scratch, both PEARL variants consistently outperform the baselines across all domains. This highlights the effectiveness of jointly learning state and action abstractions during RL. Notably, PEARL-flexible achieves the highest overall performance, demonstrating the benefits of adaptive refinement over a fixed, uniform strategy. In contrast, HyAR fails to learn effective policies in all but the Soccer domain, while MP-DQN fails across all tasks. Note that our comparison excludes the additional episodes HyAR requires to gather experience for training its continuous action embeddings, making the advantage of PEARL even more pronounced.

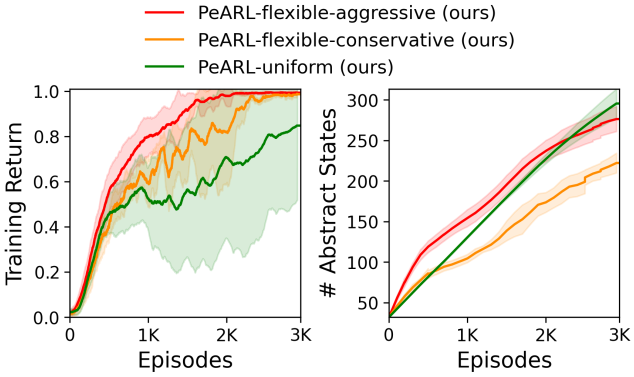


Figure 6: Comparison of training reward and state abstraction size for two PEARL-flexible variants: PEARL-flexible-aggressive, PEARL-flexible-conservative, and PEARL-uniform in the Multi-city Transport Domain.

Parsimony of abstractions Among the two PEARL variants, PEARL-flexible offers greater flexibility in controlling abstraction granularity. To investigate how abstraction granularity influences learning, we compare three configurations: two PEARL-flexible variants—aggressive vs. conservative refinement (controlled by varying the maximum number of abstract states allowed for generation per refinement)—alongside PEARL-uniform. Fig. 5 shows how these refinement strategies influence both the quality of the learned policies and the size of the resulting abstractions in the Multi-city Transport domain. The aggressively refined PEARL-flexible variant achieves the highest overall performance, demonstrating the benefits of fine-grained abstractions for precise control. In contrast, the conservatively refined variant achieves comparable performance to PEARL-uniform but results in a more compact abstraction. These

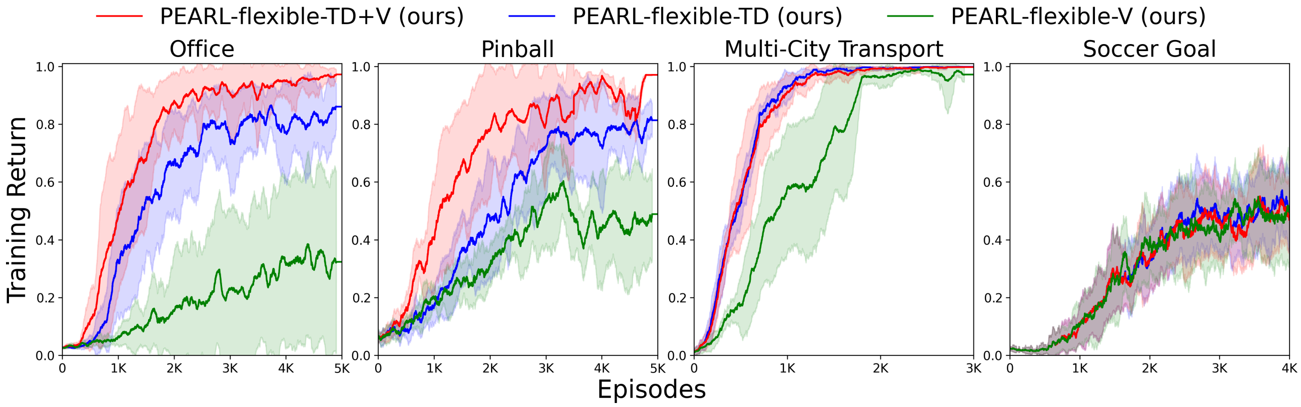


Figure 7: Comparison of PEARL's performance on all domains when using different values for annealing hyperparameter β: TD+V (β=0.02), TD (β=1.0), and V (β=0.0).

results highlight a key strength of PEARL-flexible: its ability to adjust the level of abstraction granularity based on task requirements, allowing for better tuning of the trade-off between policy performance and representational simplicity.

Impact of annealing parameter Fig. 7 shows that blending dispersion over TD-errors and values yields better performance than relying on either alone. The annealing hyperparameter provides flexibility to smoothly trade off between these signals during training.

These results support our central hypothesis that jointly learning state-action abstractions in parameterized action spaces significantly improves RL performance, enabling TD(λ) to outperform SOTA methods. PEARL-flexible learns tunable abstractions that enable a principled trade-off between computational simplicity and performance.

5 Related Work

Parameterized Actions in RL Most standard RL methods (Mnih et al. 2015; Lillicrap et al. 2015; Schulman et al. 2017; Haarnoja et al. 2018) are designed for homogeneous action spaces, handling either purely discrete or purely continuous action spaces. Moreover, their success has mostly been limited to settings with short effective horizons, where multi-step lookahead is unnecessary (Laidlaw, Russell, and Dragan 2023). Parameterized actions, which combine discrete actions with associated continuous parameters, present additional challenges they do not address. Some early approaches, such as Q-PAMDP (Masson, Ranchod, and Konidaris 2016) alternate between optimizing discrete actions and their continuous parameters. PADDPG (Hausknecht and Stone 2016) collapses all action parameters into a single continuous vector. These methods do not exploit the inherent structure of the parameterized actions (the dependency between discrete actions and their associated parameters) essential for learning effective policies.

P-DQN (Xiong et al. 2018) directly handles hybrid action spaces without relaxation or approximation by integrating a DQN (to deal with discrete actions) and a DDPG (to deal with continuous actions). However, this approach treats all action-parameters as a single joint input to the Q-network, which results in dependence of each discrete action's value on all action-parameters, not only those associated with that action. To overcome the over-parameterization problem of P-DQN, MP-DQN (Bester, James, and Konidaris 2019) extend P-DQN with a multiple-pass mechanism, splitting the action-parameter inputs to the Q-network using several passes. H-PPO (Fan et al. 2019) decomposes the action space using parallel sub-actor networks—one for discrete action selection and others for parameter learning—guided by a shared critic. HyAR (Li et al. 2022) learns a latent representation for hybrid actions via a variational autoencoder, enabling standard DRL algorithms. These methods incur added computational cost due to architectural complexity and hyperparameter sensitivity.

Abstraction Refinement in RL Coarse-to-fine RL (CRL) (Seo, Uruç, and James 2025) discretize continuous action spaces by learning a single action discretization that spans the entire state space. This is achieved by independently learning a Q-network for each action dimension. In contrast, our method learns distinct abstractions of parameterized actions conditioned on abstract states. Unlike prior top-down abstraction methods limited to discrete actions (Dadvar, Nayyar, and Srivastava 2023; Nayyar and Srivastava 2025), PEARL handles parameterized actions with continuous parameters via action abstraction and supports flexible refinement to compactly capture structure in the problem.

6 Conclusion

We introduced a unified state-action abstraction framework with algorithms for learning refinements for an understudied setting of RL with parameterized action spaces. Our contributions are: (i) a formalism for context-sensitive abstractions unifying state and action parameters, (ii) a learning-based method for refining state abstractions flexibly, and (iii) PEARL, an algorithm that jointly learns abstractions during TD(λ). A theoretical analysis of this framework is a good direction for future work.

7 Acknowledgments

We thank Shivanshu Verma for helping with an earlier version of this approach. This work was supported in part by NSF under grants IIS 2419809 and IIS 1942856.

References

Bertsekas, D. P.; et al. 2011. Dynamic programming and optimal control 3rd edition, volume ii. *Belmont, MA: Athena Scientific*, 1.

Bester, C. J.; James, S. D.; and Konidaris, G. D. 2019. Multi-pass q-networks for deep reinforcement learning with parameterised action spaces. *arXiv preprint arXiv:1905.04388*.

Corazza, J.; Aria, H. P.; Neider, D.; and Xu, Z. 2024. Expediting Reinforcement Learning by Incorporating Knowledge About Temporal Causality in the Environment. In *Proceedings of Causal Learning and Reasoning*.

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

Deng, Z.; Devic, S.; and Juba, B. 2022. Polynomial time reinforcement learning in factored state MDPs with linear value functions. In *International conference on artificial intelligence and statistics*. PMLR.

Fan, Z.; Su, R.; Zhang, W.; and Yu, Y. 2019. Hybrid actor-critic reinforcement learning in parameterized action space. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of International conference on machine learning*.

Hansen, N.; Su, H.; and Wang, X. 2024. TD-MPC2: Scalable, Robust World Models for Continuous Control. In *Proceedings of International Conference on Learning Representations*.

Hausknecht, M.; and Stone, P. 2016. Deep reinforcement learning in parameterized action space. In *Proceedings of International Conference on Machine Learning*.

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.

Kearns, M.; and Singh, S. 1998. Finite-sample convergence rates for Q-learning and indirect algorithms. *Advances in neural information processing systems*, 11.

Laidlaw, C.; Russell, S. J.; and Dragan, A. 2023. Bridging rl theory and practice with the effective horizon. In *Proceedings of Advances in Neural Information Processing Systems*.

Levy, A.; Konidaris, G.; Platt, R.; and Saenko, K. 2019. Learning multi-level hierarchies with hindsight. In *Proceedings of International Conference on Learning Representations*.

Li, B.; Tang, H.; ZHENG, Y.; HAO, J.; Li, P.; Wang, Z.; Meng, Z.; and Wang, L. 2022. HyAR: Addressing Discrete-Continuous Action Reinforcement Learning via Hybrid Action Representation. In *Proceedings of International Conference on Learning Representations*.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a unified theory of state abstraction for MDPs. *AI&M*, 1(2): 3.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Ma, Y.; Hao, X.; Hao, J.; Lu, J.; Liu, X.; Xialiang, T.; Yuan, M.; Li, Z.; Tang, J.; and Meng, Z. 2021. A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems. In *Proceeding of Advances in neural information processing systems*.

Masson, W.; Ranchod, P.; and Konidaris, G. 2016. Reinforcement learning with parameterized actions. In *Proceedings of the AAAI conference on artificial intelligence*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Murtagh, F.; and Contreras, P. 2012. Algorithms for hierarchical clustering: an overview. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 2(1): 86–97.

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. 2025. Autonomous option invention for continual hierarchical reinforcement learning and planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Oswald, J.; Srinivas, K.; Kokel, H.; Lee, J.; Katz, M.; and Sohrabi, S. 2024. Large language models as planning domain generators. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, 12: 2825–2830.

Rodriguez-Sanchez, R.; and Konidaris, G. 2024. Learning Abstract World Models for Value-preserving Planning with Options. In *Reinforcement Learning Conference*.

Roice, K.; Panahi, P. M.; Jordan, S. M.; White, A.; and White, M. 2024. A New View on Planning in Online Reinforcement Learning.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609.

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

Seo, Y.; Uruç, J.; and James, S. 2025. Continuous Control with Coarse-to-fine Reinforcement Learning. In *Proceedings of Conference on Robot Learning*.

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*.

Sutton, R. S. 1988. Learning to predict by the methods of temporal differences. *Machine learning*, 3: 9–44.

Sutton, R. S.; and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press.

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*.

Watkins, C. J. C. H.; et al. 1989. Learning from delayed rewards.

Xiong, J.; Wang, Q.; Yang, Z.; Sun, P.; Han, L.; Zheng, Y.; Fu, H.; Zhang, T.; Liu, J.; and Liu, H. 2018. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *arXiv preprint arXiv:1810.06394*.

8 Test Environments

We evaluate in domains with long-horizons and sparse rewards i.e., agents receive a positive reward only upon reaching the goal state: (i) OfficeWorld (Icarte et al. 2022; Corazza et al. 2024) (Fig. 4a): In this domain, the agent must navigate a cluttered indoor office environment to pick up mail and coffee and deliver them to designated office locations. The state space includes the agent's (x, y) position and two binary variables indicating whether it is carrying coffee or mail. The action space consists of four parameterized movement actions—one for each cardinal direction-with displacement values in the range $[0, 0.5)$. The agent automatically picks up items when at their location and drops them off when at the target office location.

(ii) Pinball (Roice et al. 2024; Rodriguez-Sanchez and Konidaris 2024) (Fig. 4b): The agent controls a small ball in a physics-based arena and must guide it into a red hole, avoiding collisions with irregularly shaped obstacles. The ball is subject to dynamic physical forces, such as bouncing off obstacles and surface resistance. The action space includes five parameterized actions: four to increase or decrease velocity in the x or y direction, and one no-op action.

(iii) Multi-city transport (Ma et al. 2021; Oswald et al. 2024) (Fig. 4d): This domain models a complex, multi-city delivery problem. The agent navigates roads within cities and uses air transport to travel between them. The objective is to retrieve a package in one city and deliver it to a destination city. The environment includes three cities, each with an airport. The agent has five parameterized actions: up, down, left, right (each parameterized by distance), and a fly action (parameterized by the destination city), which can only be executed at airports.

(iv) Robot Soccer Goal (Bester, James, and Konidaris 2019) (Fig. 4c): The task involves an agent learning to kick a ball past a keeper. Three actions are available to the agent: kick-to(x,y), shoot-goal-left(y), and shoot-goal-right(y). It terminates if the ball enters the goal, is captured by the keeper, or leaves the play area.

9 Hyperparameters

To evaluate and compare the learning performance for all the methods, we use the open-source implementations of MP-DQN[1] and HyAR[2] baselines. However, we replace their manually designed, domain-specific weight initializations with zero or randomized initializations, whichever yielded better results. We retain all other hyperparameters from their original implementations, as our extensive hyperparameter tuning did not produce better performance than the defaults.

The hyperparameters used for our methods—PEARL-flexible and PEARL-uniform—are detailed in Tables 1 and 2. Key abstraction-specific parameters include: `k_cap` and `k_cap_actions`: These define the upper bounds on the number of abstract states and abstract actions, respectively, that are eligible for refinement during each abstraction refinement phase; `max_clusters`: Specifies the number of new clusters created when refining an abstract state using flexible refinement, effectively determining how many new abstract states are generated; and `variables_to_split`: Sets the maximum number of state variables considered for uniform refinement at each step. `n_refine`: Indicates the number of episodes between successive abstraction refinement phases. β: This is the annealing hyperparameter used for computing the heterogeneity and similarity measures. In addition to these abstraction-related parameters, all standard reinforcement learning hyperparameters (e.g., learning rate, discount factor) are included to ensure reproducibility of experiments. The code, hyperparameters used, and instructions to run experiments are made open-source[3].

[1] https://github.com/cycraig/MP-DQN

[2] https://github.com/TJU-DRL-LAB/self-supervised-rl.git

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

Table 1: Hyperparameters for PEARL-flexible used with different domains

Hyperparameter	**Office**	**Pinball**	**Multi-City Transport**	**Soccer Goal**
minimum exploration ϵ_{min}	0.05	0.05	0.05	0.05
learning rate α	0.05	0.1	0.05	0.05
discount factor γ	0.99	0.999	0.99	0.99
lamda λ	0.1	0.1	0.1	0.0
maximum steps h	400	600	400	150
decay δ	0.9989	0.9997	0.9989	0.9989
n_refine n_{refine}	100	100	100	100
k_cap	2	40	10	25
k_cap_actions	3	15	10	25
max_clusters	3	4	8	20
kernel	linear	rbf	linear	linear
Annealing β	0.02	0.02	0.02	0.02

Table 2: Hyperparameters for PEARL-uniform used with different domains

Hyperparameter	**Office**	**Pinball**	**Multi-City Transport**	**Soccer Goal**
minimum exploration ϵ_{min}	0.05	0.05	0.05	0.05
learning rate α	0.05	0.1	0.05	0.05
discount factor γ	0.99	0.999	0.99	0.99
lamda λ	0.1	0.1	0.1	0.0
maximum steps h	400	600	400	150
decay δ	0.9989	0.9997	0.9989	0.9989
n_refine n_{refine}	100	100	100	100
k_cap	5	40	10	10
k_cap_actions	5	15	10	10
variables_to_split	4	2	4	2
Annealing β	0.02	0.02	0.02	0.02