Learning Context-Sensitive State and Action Abstractions for Reinforcement Learning with Parameterized Actions

Rashmeet Kaur Nayyar[1], **Naman Shah**[1,2], **Siddharth Srivastava**[1]
{rmnayyar, shah.naman, siddharths}@asu.edu

[1]**School of Computing and Augmented Intelligence, Arizona State University, AZ, USA**
[2]**Department of Computer Science, Brown University, RI, USA**

Abstract

Real-world sequential decision making problems often require parameterized action spaces that feature both, decisions regarding discrete actions and decisions about continuous action parameters governing how an action is executed. However, existing approaches exhibit severe limitations when handling such parameterized action spaces—planning algorithms require hand-crafted action models, and reinforcement learning (RL) paradigms focus on either discrete or continuous actions but not both. This paper extends the scope of RL algorithms to settings with mixtures of discrete and continuous parameterized actions through a unified view of continuous-to-discrete context-sensitive state and action abstractions. We present algorithms for online learning and flexible refinement of such abstractions during RL. Empirical results show that learning such abstractions on-the-fly enable Q-learning to significantly outperform state-of-the-art RL approaches in terms of sample efficiency across diverse problem domains with long horizons, continuous states and parameterized actions.

1 Introduction

Reinforcement learning (RL) has achieved remarkable success across a wide 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). However, most leading RL approaches (Schulman et al., 2017; Haarnoja et al., 2018; Schrittwieser et al., 2020; Hansen et al., 2023) are designed specifically for either discrete or continuous action spaces, but not both. In many real-world applications—such as autonomous driving—agents may need to choose among distinct actions such as acceleration, breaking and turning, each of which is parameterized by discrete or continuous parameters, such as breaking force and steering angle. In such cases, the agent must not only select an action but also determine its associated parameters prior to execution. These types of actions are known as *parameterized actions*.

Although there has been progress in RL with parameterized actions (Xiong et al., 2018; Bester et al., 2019; Li et al., 2021), current methods generally overlook the inherent structure within parameterized action spaces. E.g., in autonomous navigation, an agent might benefit from making fine-grained adjustments to movement parameters near obstacles, while only needing coarse-grained control in open, obstacle-free areas. Additionally, many existing approaches depend on expert-designed dense reward functions or benefit from relatively short "effective horizons" to facilitate learning (Laidlaw et al., 2023). As a result, the challenge of learning and leveraging the structure of parameterized action spaces to improve sample efficiency, particularly in long-horizon tasks, remains largely unaddressed. A comprehensive comparison of related work can be found in App. Sec. C.

This paper presents PEARL (Parameterized Extended state/action Abstractions for Reinforcement Learning), a novel algorithm for efficient decision-making with parameterized actions in long-horizon, sparse-reward settings. The key contributions of this work are: (i) A new formalization of context-sensitive abstractions that unify state abstraction framework with abstraction of continuous action parameters, (ii) A novel algorithm for generating learning-based flexible refinements of state abstraction, (iii) PEARL algorithm for concurrently learning abstractions for state and parameterized actions during Q-learning, and (iv) Empirical analysis of flexible refinement and learning state and action abstractions. Empirical evaluation on four complex long-horizon, sparse-reward domains with parameterized actions (detailed in Sec. 4), show significant performance improvements over existing RL methods and highlight PEARL's ability to learn effective policies.

2 Preliminaries

We model the problem as an episodic factored goal-oriented Markov decision process (MDP) with parameterized actions, denoted as $\mathcal{M} = \langle \mathcal{V}, \mathcal{S}, \mathcal{A}, T, R, \gamma, h, s_0, G \rangle$ (Bertsekas et al., 2011; Tarbouriech et al., 2020; Deng et al., 2022). Here, $\mathcal{V}$ is a set of state variables, each with a bounded, ordered domain $\mathcal{D}_{v_i} = \left[\mathcal{D}_{v_i}^{min}, \mathcal{D}_{v_i}^{max}\right] \subseteq \mathbb{R}$. A state $s \in \mathcal{S}$ assigns a value to every variable in $\mathcal{V}$, i.e., $s = \{v_i = x_k | v_i \in \mathcal{V} \wedge x_k \in \mathcal{D}_{v_i}\}$, and we denote the value of v_i in s as $s(v_i)$. The action set $\mathcal{A}$ contains a finite set of stochastic, parameterized actions, where each $a \in \mathcal{A}$ denoted as $a_l(a_p)$ is defined by a label a_l and a tuple of continuous parameters $a_p = \langle x_1, \ldots, x_k \rangle$, each with a bounded domain $\mathcal{D}_{x_i} \subseteq \mathbb{R}$. A grounded action $\tilde{a}_i$ assigns specific values to these parameters, and the set of all grounded actions $\tilde{\mathcal{A}}$ may be infinite due to the continuous parameter spaces. A detailed description of all the components in this formulation is presented in App. Sec. A.

Running Example Consider an AI agent in an Office World (Fig. 1a) that must collect and deliver items, such as coffee and mail, between rooms and designated offices. The state includes the agent's (x, y) position with $x, y \in [0.0, 100.0)$, and two binary variables: $c \in \{0, 1\}$ for carrying coffee, and $m \in \{0, 1\}$ for carrying mail. The agent can execute parameterized actions $\mathcal{A} = \{up(d), down(d), left(d), right(d)\}$ where parameter $d \in [0, 0.5)$ specifies the movement distance. The actions may result in stochastic displacements along orthogonal directions. Rewards are given only when items are delivered to the correct office.

In such sparse-reward, long-horizon environments, standard RL methods often suffer from poor sample efficiency and limited scalability (App. Sec. C). We review state abstraction and Conditional Abstraction Trees (CATs) (Dadvar et al., 2023) learned using the CAT+RL algorithm in App. Sec. A. These approaches, however, do not address abstraction of parameterized actions.

3 Our Approach

The central goal of this work is to develop a sample-efficient approach that leverages the structure of the parameterized action spaces to solve complex RL problems. To do so, we introduce the notion of *State and Parameterized Action-based Conditional Abstraction Trees (SPA-CATs)* (Sec. 3.1), a new flexible, learning-based abstraction refinement strategy (Sec. 3.2), and propose a novel approach—PEARL—for learning and using these abstraction trees on-the-fly during RL (Sec. 3.3).

Intuitively, SPA-CATs enable reasoning at multiple granularities by partitioning (a) continuous state regions using *State-based Conditional Abstraction Trees (S-CATs)* and (b) action parameter intervals using *Parameterized Action-based Conditional Abstraction Trees (PA-CATs)* that are conditioned on these abstract states. By conditioning action abstractions on abstract states, a SPA-CAT allows the agent to increase precision in action arguments only in states where such precision significantly impacts decision-making. We define and discuss SPA-CATs in detail in Sec. 3.1.

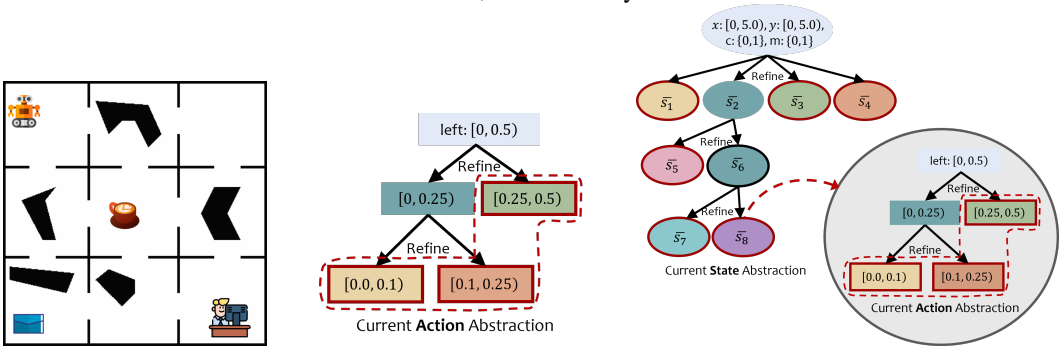
The abstraction framework introduced in this paper is, in principle, compatible with any RL algorithm, however, we focus on Q-learning for development and evaluation. PEARL defines a top-down abstraction refinement approach that starts with an initial coarse abstraction of the state and action

spaces, and proceeds through the following steps: (i) learn an abstract policy over the current state and action abstraction, (ii) refine the current state and action abstraction, and (iii) repeat steps (i) and (ii) until a policy is learned. Specifically, PeARL computes state-conditioned action abstractions dynamically, refining coarse parameter intervals into finer ones based on observed Q-value estimation errors. Sec. 3.3 discusses PEARL in detail.

For instance, consider the Office World example with parameterized actions $\{up(d), down(d), left(d), right(d)\}$, where an agent can move by a continuous distance $d \in [0, 0.5)$. In open areas, a coarse action abstraction such as "move right by *up to 0.5* units" may suffice, but it lacks precision near walls or tight corridors. PEARL addresses this by detecting regions in the state space with high value estimation errors, i.e., regions where the estimation of expected returns does not match with observed returns, and adaptively refining the action abstraction into more precise intervals (e.g., $[0, 0.25)$ and $[0.25, 0.5)$).

3.1 State and Parameterized Action Conditional Abstraction Trees (SPA-CATs)

SPA-CATs unify state and action abstractions into a single conditional abstraction tree. The state abstraction (S-CAT) follows Def. A.2, while the action abstraction is defined using the notion of PA-CATs. We first introduce abstract actions, then formally define a PA-CAT.



(a) Office World (b) PA-CAT (c) SPA-CAT

Figure 1: Illustration of PA-CAT and SPA-CAT for the office world domain.

Action Abstraction Various forms of action abstraction have been explored in the literature, including temporal actions such as options (Sutton et al., 1999; Abel et al., 2020; Machado, 2025) and discretization of continuous action spaces (Seo et al., 2024). In this work, we define an action abstraction for each parameterized action $a \in \mathcal{A}$. Furthermore, abstraction for each action is conditioned on the abstract state. For example, consider the parameterized action $a = left(d)$ with one parameter $d \in [0.0, 0.5)$. A grounded action with arguments sampled from a_p could be $\tilde{a} = left(0.2)$, which would move the agent to the left by a distance of 0.2 units. Here, possible abstract actions include: $\overline{a}_1 = left([0.0, 0.1))$ and $\overline{a}_2 = left([0.1, 0.5))$. An action abstraction $\beta : \tilde{\mathcal{A}} \to \overline{\mathcal{A}}$ maps a grounded action $\tilde{a} \in \tilde{\mathcal{A}}$ to an abstract action $\overline{a} \in \overline{\mathcal{A}}$, where $\overline{\mathcal{A}}$ represents a possible partitioning of the parameter space a_p for each grounded action $\tilde{a} \in \tilde{\mathcal{A}}$. The coarsest action abstraction for a given action $a \in \mathcal{A}$ consists of a single abstract action $\overline{a}_{init}$ (e.g. $left([0.0, 0.5))$), where each parameter $x_i \in a_p$ retains its full domain $\overline{a}_{init}(x_i) = \mathcal{D}_{x_i}$. Formally, an abstract action is defined as follows.

Definition 3.1 (Abstract Action)**.** Given a grounded action $\tilde{a} \in \tilde{\mathcal{A}}$ for a parameterized action $a \in \mathcal{A}$, an *abstract action* $\overline{a} \in \overline{\mathcal{A}}$ is defined as a function $a_l(\overline{a}_p)$ where a_l is the action label and $\overline{a}_p$ is an assignment of an interval of values $\overline{a}(x_i) \subseteq \mathcal{D}_{x_i}$ to each parameter $x_i \in a_p$.

Fig. 1b illustrates a PA-CAT for the $left(d)$ parameterized action in the office domain (Fig. 1a). Intuitively, a PA-CAT for an action represents a hierarchy of abstractions over the parameter space of an action, where: (a) the root node represents the coarsest abstraction, encompassing the entire parameter space for the action, and (b) lower-level nodes represent progressively more refined abstractions by increasing the resolution of parameters around specific arguments. The leaves of a

PA-CAT represent the current abstraction being used for the action. Note that each abstract state is associated with one PA-CAT for each action. Abstract actions from all PA-CATs associated with a state in S-CAT represent the abstract action space $\overline{\mathcal{A}}$. We formally define a PA-CAT as follows.

Definition 3.2 (Parameterized Action-based Conditional Abstraction Trees (PA-CATs))**.** A PA-CAT for a parameterized action $a \in \mathcal{A}$ is a tree-structured abstraction represented by a tuple $\langle \mathcal{N}, \mathcal{E} \rangle$, where $\mathcal{N}$ is a set of nodes, each corresponding to an abstract action, and $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ is a set of directed edges connecting parent and child abstract actions. The root node corresponds to the coarsest abstract action $\overline{a}_{init}$. An edge $e \in \mathcal{E}$ from a parent node $\overline{a}_1 \in \mathcal{N}$ to a child node $\overline{a}_2 \in \mathcal{N}$ exists if and only if the following condition holds: Let $\overline{a}_{1p} = \langle l_1, ..., l_n \rangle$ where each l_i(for $i = 1, ..., n$) can be partitioned into sub-intervals l_i^1 and l_i^2. Then, the parameter ranges of $\overline{a}_2$ must be $\overline{a}_{2p} = \langle m_1, ..., m_n \rangle$, where for each i, $m_i = l_i^1$ or l_i^2.

The leaf nodes of a PA-CAT for an action a represent the current abstraction of parameter values for a. The set of all PA-CATs (one for each action) defines an action abstraction function $\beta : \tilde{\mathcal{A}} \rightarrow \overline{\mathcal{A}}$, which maps each grounded action to the unique leaf in that action's PA-CAT that is consistent with the grounded action's parameters (e.g. the PA-CAT in Fig. 1b maps *left(0.03)* to *left([0,0.1))*. A set of PA-CATs—one for each parameterized action and abstract state—combined with a S-CAT, together form a unified abstraction structure called a SPA-CAT (see Fig. 1c). With these definitions in hand, we formally define a SPA-CAT as follows.

Definition 3.3 (State and Parameterized Action-based Conditional Abstraction Trees (SPA-CATs))**.** A SPA-CAT is a hierarchical abstraction structure defined as $\Delta = \{\Delta_{\mathcal{S}}\} \cup \{\Delta_{\overline{\mathcal{A}}}^{\overline{s}a} | \overline{s} \in \overline{\mathcal{S}}, a \in \mathcal{A}\}$, where $\Delta_{\mathcal{S}}$ is a S-CAT that partitions the concrete state space $\mathcal{S}$ into a set of abstract states $\overline{\mathcal{S}}$, and for each abstract state $\overline{s} \in \overline{\mathcal{S}}$ and parameterized action $a \in \mathcal{A}$, $\Delta_{\overline{\mathcal{A}}}^{\overline{s}a}$ is a PA-CAT that partitions the parameter space a_p of action a into a set of abstract actions, forming a subset of the overall abstract action space $\overline{\mathcal{A}}$ associated with abstract state $\overline{s}$.

3.2 Statistical Learning-Based Flexible Abstraction Refinement

A key contribution of this work is that PEARL generalizes the notion of abstraction refinement developed originally for CATs. In that work, each variable's value interval within an abstract state is uniformly split into two equal sub-intervals. As seen in Fig. 2(right), applying this uniform refinement results in orthogonal partitions of the state space. Here, the learned abstraction corresponds to navigating to pick up the mail (refer caption of Fig. 2). While straightforward, this strategy can be inefficient, as achieving fine-grained precision often requires a large number of refinements.

In this work, we introduce a novel data-driven, learning-based refinement paradigm that enables more flexible and expressive state abstractions. Unlike prior methods that use uniform splits, our approach—illustrated in Fig. 2(left)—learns adaptive boundaries to form abstract states. For example, the learned abstraction in the figure corresponds to navigating to pick up coffee. By leveraging observed data, the method groups concrete states into meaningful abstract states, resulting in more compact representations that still preserve the key distinctions necessary for effective decision-making.

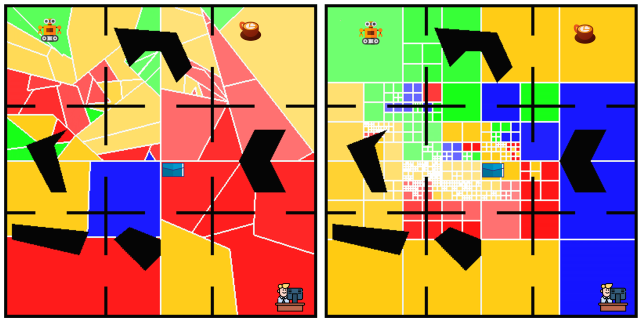


Figure 2: Visualization of learned state abstractions using flexible (left) and uniform (right) refinement strategies when the agent must pickup either coffee or mail in Office World (black lines and shapes are walls/obstacles). Colors represent actions (yellow: right, green: down, red: up, blue: left).

We now describe the flexible refinement method used to refine abstract states in a SPA-CAT. This approach uses a set of traces of the form $\langle s_i, \overline{a}_i, r_i, ... \rangle$ where s_i is the current state, $\overline{a}_i$ is the abstract

action taken, and r_i is the reward obtained in that current state. The abstraction function of the SPA-CAT maps each concrete state s to its corresponding abstract state $\overline{s}$. PEARL automatically selects a subset of abstract states to refine (see refinement phase in Sec. 3.3). The refinement process first estimates TD errors for concrete state-abstract action pairs, and then clusters these TD errors within each selected abstract state. Finally, it constructs new refined abstract states based on these clusters.

Estimating TD errors For each selected abstract state, PEARL estimates concrete Q-values (Q) and temporal-difference (TD) errors (δ) for pairs of concrete states and abstract actions. These estimates are computed over a bounded set of recently visited concrete states that belong to the selected abstract state. Rather than using the value of the next concrete state, PEARL approximates it with the estimated value of the next abstract state (see learning phase in Sec. 3.3). This results in a Q-learning-style update, but applied in the abstract space: Q-values and TD errors for concrete states are computed with respect to abstract actions, and the target uses an approximate value for the next abstract state rather than the concrete state. Let α and γ be learning rate and discount factor, respectively. Formally, this update is defined as:

$$Q_{k+1}(s,\overline{a}) = Q_k(s,\overline{a}) + \alpha \cdot \delta(s,\overline{a}), \qquad \delta(s,\overline{a}) \quad = r(s,\overline{a}) + \gamma \max_{\overline{a}'} Q_k(\overline{s}',\overline{a}') - Q_k(s,\overline{a}) \quad (1)$$

Learning refinements of abstract states Abstract states are refined by first clustering the TD errors for concrete states from that abstract state, and then training a classifier over the obtained clustering, with the objective of creating partitions that show similar TD errors. The resulting classifier represents a refinement, or partitioning of the original abstract state. Any clustering algorithm can be used to cluster concrete states based on their normalized estimated TD errors. In this paper, we use agglomerative clustering, progressively increasing the distance threshold until the number of resulting clusters falls below a predefined maximum (set by a hyperparameter). For each abstract state, PEARL trains a Support Vector Machine (SVM) classifier to distinguish between the newly formed clusters. The SVM is trained using the original concrete state features as input and cluster assignments as labels. PEARL supports different kernel functions (e.g., linear, RBF), allowing flexibility in how the decision boundaries are shaped in the input space.

3.3 PEARL Algorithm

We now present our overall approach, PEARL, for automatically learning a SPA-CAT and a policy for a given problem (App. Alg. 1). PEARL begins with an initial, coarse SPA-CAT and follows a top-down refinement strategy, incrementally refining abstractions into more fine-grained representations as needed. The initial SPA-CAT consists of a coarse S-CAT and a set of coarse PA-CATs—one for each parameterized action, defined for each abstract state of the S-CAT. By identifying abstract states and abstract actions associated with high TD error, PEARL focuses refinement efforts on regions of the abstraction that are most critical for improving policy performance.

PEARL learns a SPA-CAT and a solution policy for the input MDP $\mathcal{M}$. It initializes a coarse abstraction in line 1. This abstraction is refined while simultaneously learning a policy through two main phases: (a) a learning phase (lines 4-11), where a policy is learned for the current fixed SPA-CAT, and (b) a refinement phase (lines 12-20), where the abstraction is improved by refining the current SPA-CAT into a more fine-grained representation.

Learning phase In this phase, the agent learns an abstract policy $\pi : \overline{\mathcal{S}} \rightarrow \overline{\mathcal{A}}$ for the current SPA-CAT using tabular Q-learning over $n_{\textit{refine}}$ episodes (App. Alg. 1, lines 4-11). During each episode, the agent follows the abstract policy by executing the corresponding abstract action in each abstract state, continuing until it reaches a new abstract state or the episode ends (lines 7-8). The SPA-CAT representation of abstract actions enables the joint optimization of both action selection and parameter values. Each abstract action is executed by uniformly sampling a grounded action from its associated parameter ranges. During learning, PEARL collects a set of execution traces of the form $\langle s_i, \overline{a}_i, r_i, ... \rangle$ where s_i is the current state, $\overline{a}_i$ is the abstract action taken, and r_i is the

reward received in that state (line 9). We apply the standard Q-learning updates for Q-values and TD errors over abstract states and actions as follows (lines 10-11):

$$Q_{k+1}(\overline{s},\overline{a}) = Q_k(\overline{s},\overline{a}) + \alpha \cdot \delta(\overline{s},\overline{a}), \qquad \delta(\overline{s},\overline{a}) \quad = r(\overline{s},\overline{a}) + \gamma \max_{\overline{a}'} Q_k(\overline{s}',\overline{a}') - Q_k(\overline{s},\overline{a}) \quad (2)$$

Here, $r(\overline{s},\overline{a})$ is the cumulative reward received in the abstract state $\overline{s}$ after executing the abstract action $\overline{a}$; α and γ are learning rate and discount factor, respectively. A log of temporal-difference (TD) errors is maintained for use during the refinement process (line 11).

Refinement phase After every n_{refine} episodes, PEARL enters the refinement phase to update the SPA-CAT (App. Alg. 1, lines 12-20). This phase begins by analyzing the log of previously computed TD errors to identify abstract states and abstract actions with high variability in their TD errors, as this indicates instability in value estimates (line 13). PEARL then refines the selected abstract states using the chosen refinement strategy—either flexible (Sec. 3.2) or uniform (Dadvar et al., 2023). For abstract actions, PEARL applies the uniform refinement strategy, which increases precision across all parameter ranges associated with the selected actions (lines 14-19). The updated SPA-CAT is then used to continue the learning phase.

4 Empirical Results

We evaluate PeARL's performance in three challenging stochastic domains featuring continuous state spaces and parameterized action spaces along with sparse rewards and long-horizon tasks, posing significant challenges for RL algorithms. Further details including additional evaluations, analysis, and detailed hyperparameter settings are provided in App. Sec. D.

Test environments We evaluate in four long-horizon domains, each featuring 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. 1a) (ii) Pinball (Roice et al., 2024; Rodriguez-Sanchez & Konidaris, 2024) (App. Fig. 4b), (iii) Multi-city transport (Ma et al., 2021; Oswald et al., 2024) (App. Fig. 4d), and (iv) Robot Soccer Goal (Bester et al., 2019) (App. Fig. 4c). Detailed descriptions and illustrations of these environments are provided in App. Sec. D.

Qualitative metrics and baseline selection Most existing 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., 2017) approaches do not handle parameterized action spaces, making them unsuitable for baseline comparisons. We therefore compare PEARL with two relevant baselines: (i) MP-DQN (Bester et al., 2019): Extend P-DQN (Xiong et al., 2018) by combining DQN and DDPG while addressing P-DQN's over-parameterization problem through multi-pass processing, and (ii) PA-DDPG (Hausknecht & Stone, 2015), which treats parameterized actions as continuous vectors and applies DDPG directly to the relaxed action space. We evaluate agents by assessing cumulative average return during training and success rate of the learned greedy policy. For PeARL, we assess two variants: PEARL-flexible and PEARL-uniform, which differ in how they refine state abstractions. Reported results include episodes used to learn both state and action abstractions.

4.1 Analysis of the results

Our evaluation focuses on three key aspects: (1) improvements in sample-efficiency, (2) the quality of the learned policies, and (3) the size of the abstractions generated. Fig. 3 shows the performance of all methods, with training episodes on the x-axis and cumulative return (training) and success probability (evaluation) on the y-axis. We also show the sizes of the state abstractions learned by PEARL. Results report mean and standard deviation over 10 independent trials. Fig. 3 shows that PEARL performs significantly better compared to the baselines across all domains, indicating the

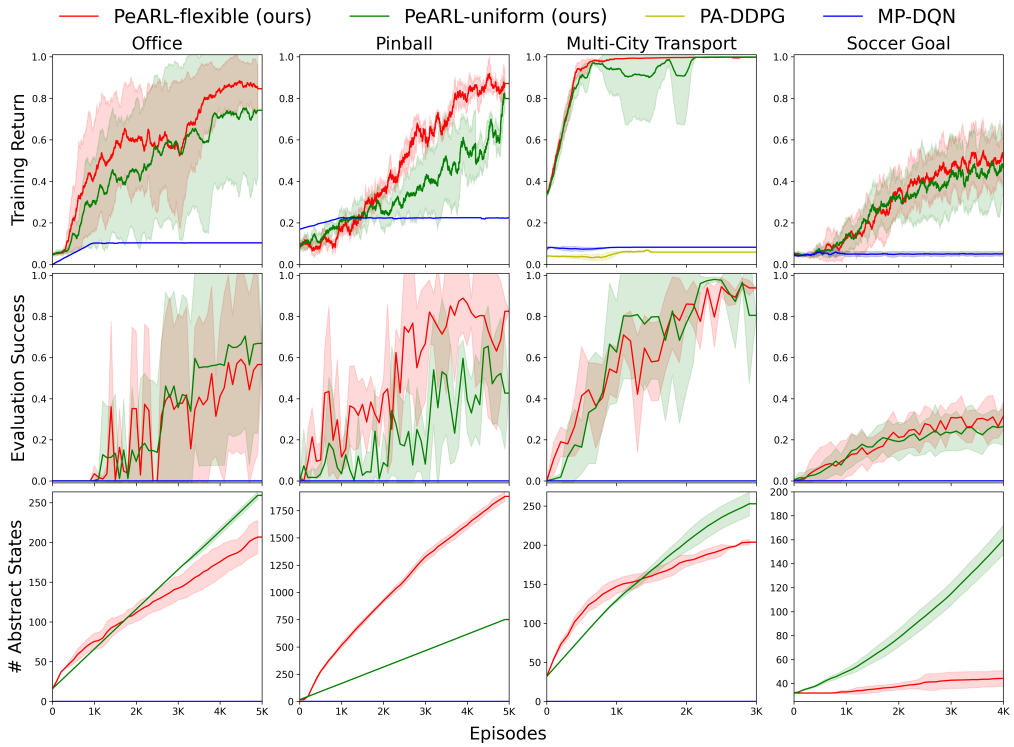


Figure 3: Comparison of PeARL-flexible and PeARL-uniform with MP-DQN and PA-DDPG in four domains: Office World, Pinball, Multi-City Transport, and Soccer Goal. Mean and Standard deviations are reported across 10 independent trials.

value of learning state and action abstractions simultaneously during Q-learning for parameterized action and continuous state spaces, especially for long-horizon problems.

As illustrated in Fig. 3, the PEARL-flexible variant consistently outperforms other approaches across all evaluated domains, followed closely by PEARL-uniform. In most environments, PEARL-flexible matches or exceeds the performance of PEARL-uniform while maintaining a more compact abstraction. Specifically, in the Office World, Pinball, and Multi-City Transport domains, PEARL-flexible achieves the highest average returns, demonstrating clear gains in sample-efficiency and policy effectiveness. In the Soccer Goal domain, both PEARL-flexible and PEARL-uniform show comparable performance, indicating that either abstraction strategy is sufficient in this environment.

A closer look at the abstraction sizes reveals that PEARL-flexible learns significantly more compact representations in three of the four domains—Office World, Multi-City Transport, and Soccer Goal—without compromising performance. In the Pinball domain, the more aggressive refinement strategy employed by PEARL-flexible leads to superior policy performance while producing higher abstraction granularity. This suggests that fine-grained abstractions are particularly beneficial in environments requiring precise control and spatial reasoning.

We further analyze how abstraction granularity affects PEARL's performance in the Pinball domain by comparing two PEARL-flexible variants—aggressive vs. conservative refinement—alongside PEARL-uniform and provide our analysis in App. Sec. D.

5 Conclusion

We introduced a unified framework for learning abstractions in 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, and (iii) PEARL, an algorithm that jointly learns abstractions during Q-learning. Experiments show PEARL outperforms existing methods, showing strong sample efficiency and effective policy learning in challenging environments.

References

David Abel, Nate Umbanhowar, Khimya Khetarpal, Dilip Arumugam, Doina Precup, and Michael Littman. Value preserving state-action abstractions. In *International Conference on Artificial Intelligence and Statistics*, pp. 1639–1650. PMLR, 2020.

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

Craig J Bester, Steven D James, and George D Konidaris. Multi-pass q-networks for deep reinforcement learning with parameterised action spaces. *arXiv preprint arXiv:1905.04388*, 2019.

Jan Corazza, Hadi Partovi Aria, Daniel Neider, and Zhe Xu. Expediting reinforcement learning by incorporating knowledge about temporal causality in the environment. In *Causal Learning and Reasoning*, pp. 643–664. PMLR, 2024.

Mehdi Dadvar, Rashmeet Kaur Nayyar, and Siddharth Srivastava. Conditional abstraction trees for sample-efficient reinforcement learning. In *Uncertainty in Artificial Intelligence*, pp. 485–495. PMLR, 2023.

Zihao Deng, Siddartha Devic, and Brendan Juba. Polynomial time reinforcement learning in factored state mdps with linear value functions. In *International conference on artificial intelligence and statistics*, pp. 11280–11304. PMLR, 2022.

Zhou Fan, Rui Su, Weinan Zhang, and Yong Yu. Hybrid actor-critic reinforcement learning in parameterized action space. *arXiv preprint arXiv:1903.01344*, 2019.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. Pmlr, 2018.

Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*, 2023.

Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015.

Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73:173–208, 2022.

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

Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. *arXiv preprint arXiv:1712.00948*, 2017.

Boyan Li, Hongyao Tang, Yan Zheng, Jianye Hao, Pengyi Li, Zhen Wang, Zhaopeng Meng, and Li Wang. Hyar: Addressing discrete-continuous action reinforcement learning via hybrid action representation. *arXiv preprint arXiv:2109.05490*, 2021.

Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. *AI&M*, 1(2):3, 2006.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Yi Ma, Xiaotian Hao, Jianye Hao, Jiawen Lu, Xing Liu, Tong Xialiang, Mingxuan Yuan, Zhigang Li, Jie Tang, and Zhaopeng Meng. A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems. *Advances in neural information processing systems*, 34:23609–23620, 2021.

Marlos C Machado. Representation-driven option discovery in reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 28705–28705, 2025.

Warwick Masson, Pravesh Ranchod, and George Konidaris. Reinforcement learning with parameterized actions. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Rashmeet Kaur Nayyar and Siddharth Srivastava. Autonomous option invention for continual hierarchical reinforcement learning and planning. 2025.

James Oswald, Kavitha Srinivas, Harsha Kokel, Junkyu Lee, Michael Katz, and Shirin Sohrabi. Large language models as planning domain generators. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pp. 423–431, 2024.

R Rodriguez-Sanchez and G Konidaris. Learning abstract world models for value-preserving planning with options. *Reinforcement Learning Journal*, 2024.

Kevin Roice, Parham Mohammad Panahi, Scott M Jordan, Adam White, and Martha White. A new view on planning in online reinforcement learning. 2024.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Younggyo Seo, Jafar Uruç, and Stephen James. Continuous control with coarse-to-fine reinforcement learning. *arXiv preprint arXiv:2407.07787*, 2024.

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

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Jean Tarbouriech, Evrard Garcelon, Michal Valko, Matteo Pirotta, and Alessandro Lazaric. No-regret exploration in goal-oriented reinforcement learning. In *International Conference on Machine Learning*, pp. 9428–9437. PMLR, 2020.

Zizhao Wang, Caroline Wang, Xuesu Xiao, Yuke Zhu, and Peter Stone. Building minimal and reusable causal state abstractions for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 15778–15786, 2024.

Christopher John Cornish Hellaby Watkins et al. Learning from delayed rewards. 1989.

Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *arXiv preprint arXiv:1810.06394*, 2018.

Appendix

A Preliminaries

We model the problem as an episodic factored goal-oriented Markov decision process (MDP) with parameterized actions, denoted as $\mathcal{M} = \langle \mathcal{V}, \mathcal{S}, \mathcal{A}, T, R, \gamma, h, s_0, G \rangle$ (Bertsekas et al., 2011; Tarbouriech et al., 2020; Deng et al., 2022). Here, $\mathcal{V}$ is a set of state variables, where each variable $v \in \mathcal{V}$ has a bounded, ordered domain $\mathcal{D}_{v_i} = \left[\mathcal{D}_{v_i}^{min}, \mathcal{D}_{v_i}^{max}\right] \subseteq \mathbb{R}$, with $\mathcal{D}_{v_i}^{min}$ and $\mathcal{D}_{v_i}^{max}$ denoting the minimum and maximum values the variable can take. The state space $\mathcal{S}$ consists of factored states. Each 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 continuous parameters where each parameter x_i has a bounded, ordered domain $\mathcal{D}_{x_i} \subseteq \mathbb{R}$. A grounded action $\tilde{a}_i$ assigns argument values to these parameters from their respective domains. The set of all possible grounded actions is denoted $\tilde{\mathcal{A}}$, and may be infinite due to continuous parameters.

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

The objective is to solve MDP $\mathcal{M}$ by finding a policy $\pi_{\mathcal{M}} : \mathcal{S} \to \tilde{\mathcal{A}}$ that reaches a goal state 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 & Barto, 1998).

Running Example Consider an AI agent in an Office World (Fig. 1a) that must collect and deliver items, such as coffee and mail, between rooms and designated offices. The state includes the agent's (x, y) position with $x, y \in [0.0, 100.0)$, and two binary variables: $c \in \{0, 1\}$ for carrying coffee, and $m \in \{0, 1\}$ for carrying mail. The agent can execute parameterized actions $\mathcal{A} = \{up(d), down(d), left(d), right(d)\}$ where parameter $d \in [0, 0.5)$ specifies the movement distance. The actions may result in stochastic displacements along orthogonal directions. Rewards are given only when items are delivered to the correct office.

In such environments with sparse rewards and long horizons, standard RL algorithms often suffer from poor sample efficiency and limited scalability (see Sec. C for detailed related work).

State Abstraction Abstraction has been recognized as essential for scalability in long horizon, sparse reward settings (Li et al., 2006; Wang et al., 2024). A state abstraction is a mapping $\alpha : \mathcal{S} \to \overline{\mathcal{S}}$ 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 $\mathcal{S}$. Given a set of variables $\mathcal{V}$, the value of a variable $v_i \in \mathcal{V}$ in an abstract state $\overline{s}$ is denoted $\overline{s}(v_i)$. The coarsest state abstraction, denoted $\overline{s}_{\textit{init}}$, assigns each variable its full domain: $\overline{s}_{\textit{init}}(v_i) = \mathcal{D}_{v_i}$, mapping each concrete state to a single abstract state. Formally,

Definition A.1 (Abstract State)**.** Given a set of variables $\mathcal{V}$ and corresponding domains $\mathcal{D}_{v_i}$ for each $v_i \in \mathcal{V}$, an *abstract state* $\overline{s} \in \overline{\mathcal{S}}$ is an assignment of a subset of values $\overline{s}(v_i) \subseteq \mathcal{D}_{v_i}$ to each v_i.

In order to reason at different levels of granularity, state abstractions can be organized hierarchically in a Conditional Abstraction Tree (CAT) (Dadvar et al., 2023). Intuitively, in this structure, the root node represents the coarsest abstract state. As we move down the tree, each level introduces more refined abstract states, with increased precision in the variables that require greater detail. The CAT+RL algorithm can be used to learn CATs during Q-learning, however, is limited to domains with discrete actions and does not address abstraction of parameterized actions. Formally, CATs are defined as follows.

Definition A.2 (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 $\overline{s}_1 \in \mathcal{N}$ to a child abstract state $\overline{s}_2 \in \mathcal{N}$ exists iff $\overline{s}_2$ can be obtained by splitting atleast one of the variable intervals in $\overline{s}_1$ at most once. Leaf nodes represent the current abstract state space $\overline{\mathcal{S}}$. Δ induces a state abstraction $\alpha : \mathcal{S} \to \overline{\mathcal{S}}$ mapping each state $s \in \mathcal{S}$ to the abstract state $\overline{s} \in \overline{\mathcal{S}}$ represented by the unique leaf containing s.

B PEARL algorithm

Algorithm 1: PEARL

Input: MDP $\mathcal{M} = \langle \mathcal{V}, \mathcal{S}, \mathcal{A}, T, R, \gamma, h \rangle$
Output: Policy π for MDP $\mathcal{M}$ and SPA-CAT Δ

Initialize SPA-CAT Δ and Qtable $\overline{Q}_{\overline{s},\overline{a}}$
Initialize TD-error logs $\overline{\Gamma}_{\overline{s},\overline{a}}$ and $\Gamma_{s,\overline{a}}$
for *episode* $= 1 : n_{epi}$ **do**
    `// Learning phase`
    $s \leftarrow$ reset()
    $\tau \leftarrow \emptyset$
    for *step* $= 1 : h$ **do**
        $\overline{a} \leftarrow \pi(\overline{Q}_{\overline{s},\overline{a}}, \overline{s})$
        $\overline{s}', \overline{r}, \{s_i, \overline{a}_i, r_i, \ldots, s_k\} \leftarrow$ execute$(s, \overline{a})$
        τ.add$(\{s_i, \overline{a}_i, r_i, \ldots, s_k\})$
        $\overline{Q}_{\overline{s},\overline{a}} \leftarrow$ updateQvalue$(\overline{s}, \overline{a}, \overline{s}', \overline{r})$
        $\overline{\Gamma}_{\overline{s},\overline{a}}$.add(computeTDerror$(\overline{s}, \overline{a}, \overline{s}', \overline{r})$)
    `// Refinement phase`
    if *episode* $\bmod n_{refine} = 0$ **then**
        $\bar{\mathcal{S}}_{ref}, \bar{\mathcal{A}}_{ref} \leftarrow$ findImprecise$(\overline{\Gamma}_{\overline{s},\overline{a}})$
        if *refinement* == *flexible* **then**
            $\Gamma_{s,\overline{a}} \leftarrow$ estimateConcreteTDError$(\tau, \bar{\mathcal{S}}_{ref}, \bar{\mathcal{A}}_{ref}, \Delta)$
            $\mathcal{C} \leftarrow$ cluster$(\bar{\mathcal{S}}_{ref}, \Gamma_{s,\overline{a}})$
            $\Delta \leftarrow$ refine$(\mathcal{C}, \bar{\mathcal{S}}_{ref}, \bar{\mathcal{A}}_{ref})$
        else
            $\Delta \leftarrow$ refine$(\bar{\mathcal{S}}_{ref}, \bar{\mathcal{A}}_{ref})$
        Reinitialize $\overline{\Gamma}_{\overline{s},\overline{a}}$ and $\Gamma_{s,\overline{a}}$
return π, Δ

C Related Work

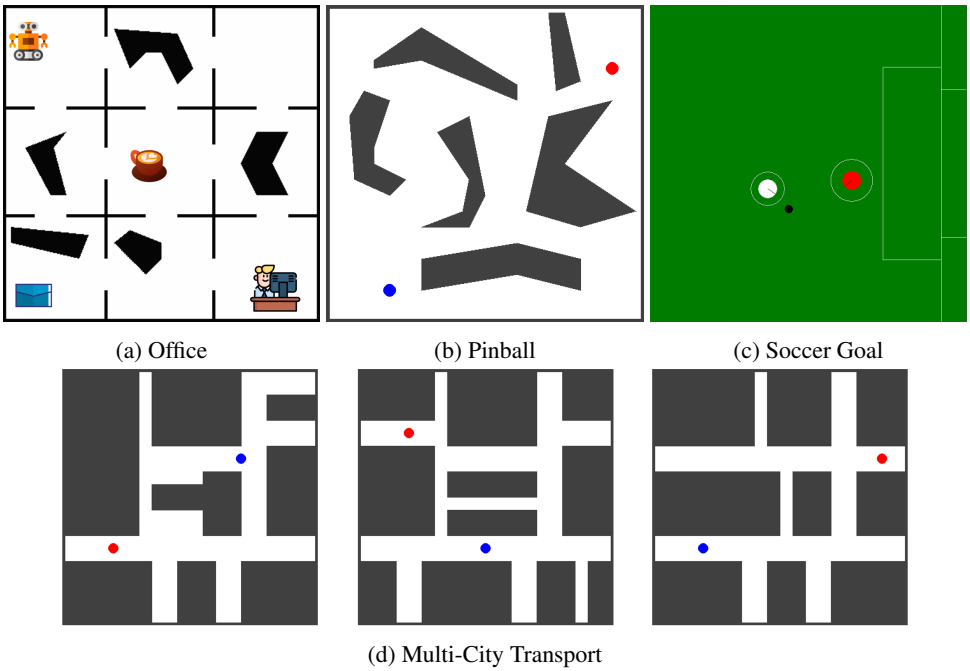
Parameterized Actions in RL Most standard RL approaches (Mnih et al., 2015; Lillicrap et al., 2015; Schulman et al., 2017; Haarnoja et al., 2018) are designed for homogeneous action spaces, handling either purely discrete or purely continuous action spaces. Moreover, their success has mostly been limited to settings with short effective horizons, where multi-step lookahead is unnecessary (Laidlaw et al., 2023). Parameterized action spaces, which combine discrete actions with associated continuous parameters, present additional challenges they do not address. Some early approaches, such as Q-PAMDP (Masson et al., 2016) alternate between optimizing discrete actions and their continuous parameters. PADDPG (Hausknecht & Stone, 2015) collapses the parameterized action space into a single continuous vector. These approaches do not exploit the inherent structure of the parameterized action spaces—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-paramaterisation problem of P-DQN, MP-DQN (Bester et al., 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., 2021) 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 et al., 2024) 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 et al., 2023; Nayyar & Srivastava, 2025), PEARL handles parameterized actions via action abstraction and supports flexible refinement to compactly capture problem structure.

D Empirical Results



(a) Office (b) Pinball (c) Soccer Goal

(d) Multi-City Transport

Figure 4: (a) A small, dynamic blue ball that needs to be manouvered into a red hole, avoiding collisions with irregularly shaped obstacles. (b) Maps for three different cities with package locations marked by red circles, connected only via airports shown as blue circles.

Test environments We evaluate in three long-horizon domains, each featuring 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. 1a): 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. The horizon is

800 steps. (ii) Pinball (Roice et al., 2024; Rodriguez-Sanchez & Konidaris, 2024) (Fig. 4b): In this domain, the agent controls a small ball in a physics-based arena and must guide it into a red hole. 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. The horizon is 800 steps. (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. The horizon is 1000 steps. (iv) Robot Soccer Goal (Bester et al., 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) for a maximum of 150 steps.

D Results and Analysis

In addition to the evaluation from the main paper, we further investigate how different abstraction refinement strategies affect the performance of PEARL in the Pinball domain. Specifically, we compare two variants of PEARL-flexible: one that uses aggressive refinement, resulting in finer-grained abstractions, and another that uses conservative refinement, leading to coarser abstractions. We also evaluate these alongside the PEARL-uniform variant. This comparison allows us to assess how the granularity of abstraction influences both learning performance and the resulting abstraction size.

As shown in Fig. 5, the aggressively refined PEARL-flexible variant achieves the highest performance among all methods. In contrast, the conservatively refined PEARL-flexible yields performance that is comparable to PEARL-uniform but with a more compact abstraction. This comparison highlights a key strength of PEARL-flexible: its ability to adaptively tune the level of abstraction to balance performance and abstraction complexity. When refined aggressively, the abstraction captures fine-grained distinctions that lead to improved control and higher sample-efficiency. On the other hand, a more conservative refinement retains a compact representation while still maintaining competitive performance, which may be beneficial in settings where computational or memory constraints are a concern.

Overall, these results reinforce our hypothesis: PEARL not only improves sample-efficiency but also produces domain-adaptive abstractions that support strong RL performance across a range of environments. The results demonstrate the abstraction learned by PEARL-flexible is not only effective but also tunable—enabling practitioners to trade off between computational simplicity and performance depending on the demands of the task.

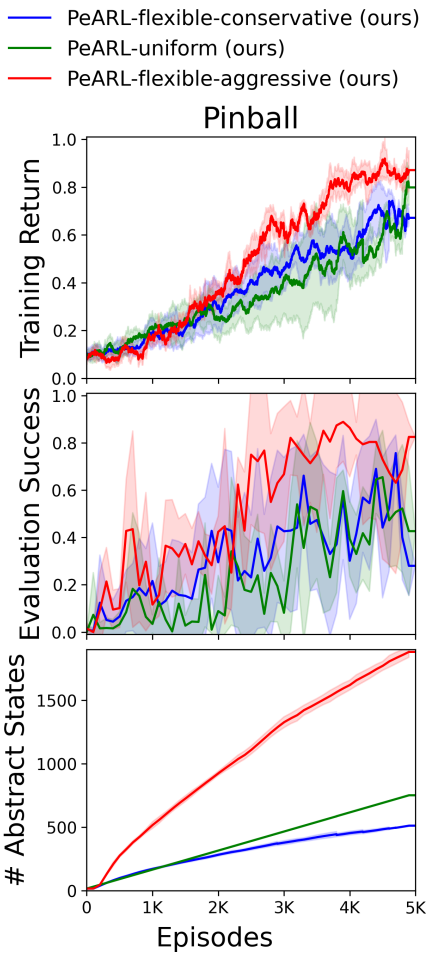


Figure 5: Comparison of two variants of PeARL-flexible: PeARL-flexible-aggressive and PeARL-flexible-conservative with PeARL-uniform in the Pinball Domain.

D Hyperparameters

For the MP-DQN and PADDPG baselines, we use the default hyperparameters provided in their open-source implementation [1]. 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, effectively determining how many new abstract states are generated; and `variables_to_split`: Sets the maximum number of state variables considered for refinement at each step. `n_refine`: Indicates the number of episodes between successive abstraction refinement phases. 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.

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	0.05	0.05	0.05
`discount factor` γ	0.99	0.999	0.99	0.99
`lamda` λ	0.1	0.1	0.1	0.1
`maximum episodes` n	5000	5000	3000	4000
`maximum steps` h	800	800	1000	150
`decay` δ	0.9989	0.9997	0.9989	0.9989
`n_refine` n_{refine}	100	100	100	100
`k_cap`	5	40	10	25
`k_cap_actions`	5	15	10	25
`max_clusters`	4	4	8	20
`kernel`	linear	rbf	linear	linear

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	0.05	0.05	0.05
`discount factor` γ	0.99	0.999	0.99	0.99
`lamda` λ	0.1	0.1	0.1	0.1
`maximum episodes` n	5000	5000	3000	4000
`maximum steps` h	800	800	1000	150
`decay` δ	0.9989	0.9996	0.9989	0.9989
`n_refine` n_{refine}	100	100	100	100
`k_cap`	5	15	10	10
`k_cap_actions`	5	15	10	10
`variables_to_split`	4	2	4	2

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