LICORICE: LABEL-EFFICIENT CONCEPT-BASED INTERPRETABLE REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in reinforcement learning (RL) have predominantly leveraged neural network-based policies for decision-making, yet these models often lack interpretability, posing challenges for stakeholder comprehension and trust. Concept bottleneck models offer an interpretable alternative by integrating human-understandable concepts into neural networks. However, a significant limitation in prior work is the assumption that annotations for these concepts are readily available during training. For RL, this requirement necessitates continuous real-time concept annotation. This reliance either places a significant burden on human annotators or incurs substantial costs in API queries and inference time when employing automated labeling methods. To overcome this limitation, we introduce a novel training scheme that enables RL algorithms to efficiently learn a concept-based policy by only querying annotators to label a small set of data. Our algorithm, LICORICE, involves three main contributions: interleaving concept learning and RL training, using a concept ensemble to actively select informative data points for labeling, and decorrelating the concept data with a simple strategy. We show how LICORICE reduces human labeling efforts to 500 or fewer concept labels in three environments and 5000 in another complex environment at minimal or no cost to performance. We also explore the use of VLMs as automated concept annotators, finding them effective in some cases but challenging in others. This work significantly reduces the annotation burden for interpretable RL, making it more practical for real-world applications where transparency is crucial.

1 INTRODUCTION

In reinforcement learning (RL), agents are tasked with learning a *policy*, a rule that makes sequential, reactive decisions in complex environments. In recent RL work, agents typically represent the policy as a neural network, as such representations tend to lead to high performance (Mirhoseini et al., 2021). However, this choice can come at a cost: such policies are challenging for stakeholders to interpret — particularly when the network inputs are also complex, such as high-dimensional sensor data. This opacity can pose a significant hurdle, especially in applications where understanding the rationale behind decisions is critical, such as healthcare (Yu et al., 2021) or finance (Liu et al., 2022). In such applications, decisions can have significant consequences, so it is essential for stakeholders to fully grasp the reasoning behind actions in order to confidently adopt or collaborate on a policy.

To address interpretability concerns in supervised learning, recent works have integrated human-understandable concepts into neural networks through concept bottleneck models (Koh et al., 2020; Espinosa Zarlenga et al., 2022). These models insert a bottleneck layer whose units correspond to interpretable concepts, ensuring that the final decisions depend on these concepts instead of on opaque raw inputs. By training the model both to have high task accuracy and to accurately match experts' concept labels, these models learn a high-level concept-based representation that is simultaneously meaningful to humans and useful for machine learning tasks. As an example, a concept-based explanation for a bird classification task might include a unit that encodes the bird's wing color.

More recently, these techniques have been applied to RL by incorporating a concept bottleneck in the policy (Grupen et al., 2022; Zabounidis et al., 2023), so that the actions taken by the agent are a function of the human-understandable concepts. However, a significant challenge emerges when we consider the practical implementation of this method: past work assumes that concept annotations

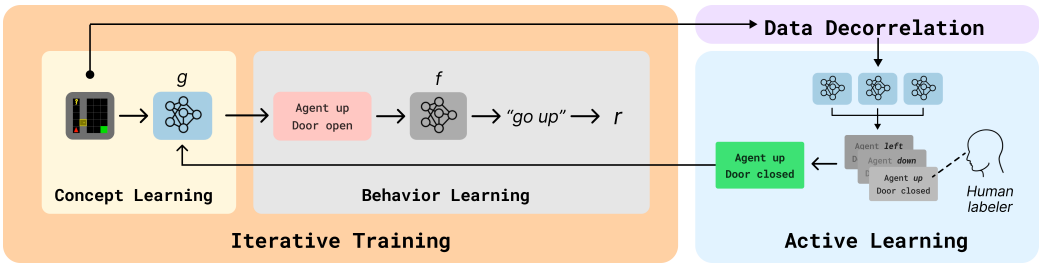

Figure 1: **LICORICE overview.** In concept-based RL, the policy includes a concept bottleneck to map from states to concepts with g, and then from concepts to (distributions over) actions with f. During training, LICORICE addresses concept label efficiency concerns with three key components: i) iterative training, ii) data decorrelation, and iii) active learning.

are readily available during the RL training process. To learn a mapping from states to concepts, an RL agent requires concept information for every state and action it encounters during training, which can often be measured in millions or billions of state-action pairs. In reality, many real-world domains of interest—such as autonomous driving—are not accompanied by high-level concepts to support human-understandable decision-making. This requirement presents major hurdles in practical implementation. On one hand, relying on human labelers for such vast datasets is impractical, risking errors due to fatigue (Marshall & Shipman, 2013) and potentially biasing the model training process. On the other hand, using vision language models (VLMs) for automated concept extraction (Oikarinen et al., 2023), while alleviating the human bottleneck, introduces significant API or inference costs that can be prohibitive given the scale of typical RL training sets.

In this work, we address this challenge with LICORICE (**L**abel-efficient **I**nterpretable **CO**ncept-based **R**e**I**nfor**CE**ment learning), a novel training paradigm designed to minimize the number of concept annotation queries while maintaining high task performance. Figure 1 illustrates our algorithm. LICORICE tackles three key challenges. First, it addresses the problem of concept learning on off-policy or outdated data. If concepts are learned from data collected by a random policy, the concept distribution may not reflect the distribution under an optimal policy. To ensure that concept learning occurs on more recent and on-policy data, LICORICE interleaves concept learning and RL training through *iterative training*: it alternately freezes the network layers corresponding to either the concept learning part or the decision-making part. Second, LICORICE addresses the problem of limited training data diversity that occurs when an agent interacts with the environment, thereby generating sequences of highly similar, temporally correlated data points. To tackle this issue, LICORICE implements a *data decorrelation* strategy to produce a more diverse set of training samples. Third, LICORICE addresses the inefficient use of annotation effort, where labeled samples may provide redundant information. To resolve this problem, we employ *disagreement-based active learning* using a concept ensemble to select the most informative data points for labeling.

To evaluate the effectiveness of LICORICE, we conduct experiments in two scenarios—perfect human annotation and VLM-based annotation—on four environments with image input: an image-based version of CartPole, two Minigrid environments, and Atari Boxing. First, under the assumption of perfect human annotation, we show that LICORICE yields both higher concept accuracy and higher reward while requiring fewer annotation queries compared to baseline methods. Second, we find that VLMs can indeed serve as concept annotators for some, but not all, of the environments.

In general, our contributions are as follows:

- To the best of our knowledge, we are the first to investigate the problem of a limited concept annotation budget for interpretable RL. We introduce LICORICE, a novel training scheme that enables label efficient learning of concept-based RL policies.
- We conduct extensive experiments to show the effectiveness of LICORICE across four environments with varying budget constraints.
- We study the use of VLMs as automated concept annotators for concept-based RL, demonstrating their effectiveness in certain environments while highlighting challenges in others.

2 PRELIMINARIES

Reinforcement Learning In RL, an agent learns to make decisions by interacting with an environment (Sutton & Barto, 2018). The environment is commonly modeled as a Markov decision process (Puterman, 2014), consisting of the following components: a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a state transition function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ that indicates the probability of transitioning from one state to another given an action, a reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ that assigns a reward for each state-action-state transition, and a discount factor $\gamma \in [0, 1]$ that determines the present value of future rewards. The agent learns a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, which maps states to distributions over actions with the aim of maximizing the expected cumulative discounted reward. We evaluate a policy via its value function, which is defined as $V^\pi(s) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s]$. The ultimate aim in RL is to determine the optimal policy, π^*, through iterative refinement based on environmental feedback.

Concept Policy Models Concept-based explanations have emerged as a common paradigm in explainable AI (Poeta et al., 2023). They explain a model's decision-making process through human-understandable attributes and abstractions. In supervised learning, concept bottleneck models (Koh et al., 2020) implement this approach using two key functions: a concept predictor $g : X \to C$, mapping inputs to interpretable concepts, and a label predictor $f : C \to Y$, mapping the concept predictions to a downstream task space, such as labels for classification. As a result, the prediction takes the form $\hat{y} = f(g(x))$, where the input x influences the output solely through the bottleneck $\hat{c} = g(x)$. In RL, we insert a concept bottleneck layer into the policy network:

$$\pi(s) = f(g(s)),$$

such that π maps from states $s \in \mathcal{S}$ to concepts $c \in C$ to actions $a \in \mathcal{A}$ (Grupen et al., 2022; Zabounidis et al., 2023). This setup allows the policy to base its decisions on understandable and meaningful features. As a result, we can use any RL algorithm that can be modified to include an additional loss function for concept prediction.

Training Concept Policy Models Training these models requires a dataset of state-concept-action triplets $(s, c, a) \in \mathcal{S} \times C \times \mathcal{A}$. The functions f and g are typically implemented as neural networks, with their parameters collectively defined as θ. Previous work (Zabounidis et al., 2023) simply combines the concept prediction loss $L^C(\theta)$ and RL loss $L^{\text{RL}}(\theta)$:

$$L(\theta) = L^{\text{RL}}(\theta) + L^C(\theta),$$

where the exact definitions of $L^{\text{RL}}(\theta)$ and $L^C(\theta)$ depend on the choice of RL algorithm and concept learning task. The objective is to find the optimal parameters θ^* that minimize this combined loss function. However, this approach requires continuous access to ground-truth concepts for training f, which may not always be feasible or desirable in practical RL scenarios.

3 LICORICE

As we have mentioned, the standard way of training concept-based RL assumes continuous access to an oracle to provide concept labels. However, this assumption is problematic due to the large annotation cost incurred by human or automated labeling efforts. To reduce the number of concept labels required for concept-based RL, we propose LICORICE, a novel algorithm for interpretable RL consisting of three main components: iterative training, data decorrelation, and disagreement-based active learning. The full pseudocode is in Algorithm 1.

Iterative Training A key challenge in concept-based RL under limited labeling resources is the changing distribution of visited states and their associated concepts as the agent's policy improves. Consider an MDP where states are indexed, and with a random (initial) policy, the agent tends to visit small-index states near the initial state, encountering only the concepts relevant to those states. However, as the policy improves, the agent explores higher-index states, leading to a shift in both the state and concept visitation distribution. If we use all of our queries at the beginning of training, we risk training the model only on the concepts associated with small-index states from the random policy, potentially missing important concepts that emerge as the agent explores more of the environment. We therefore propose iterative training to enable LICORICE to progressively refine its understanding of concepts as the policy improves. Iterative training consists of two parts:

Algorithm 1 LICORICE (**L**abel-efficient **I**nterpretable **CO**ncept-based **R**e**I**nfor**CE**ment learning)

1: **Input:** Total budget B, number of iterations M, sample acceptance threshold p, ratio τ for active learning, batch size for querying b, number of concept models N to ensemble
2: **Initialize:** training set $\mathcal{D}_{\text{train}} \leftarrow \emptyset$, and validation set $\mathcal{D}_{\text{val}} \leftarrow \emptyset$
3: **for** $m = 1$ **to** M **do**
4: Allocate budget for iteration m: $B_m \leftarrow \frac{B}{M}$
5: **while** $|\mathcal{U}_m| < \tau \cdot B_m$ **do**
6: Execute policy π_m to collect unlabeled data $\mathcal{U}_m$ using acceptance rate p
7: **end while**
8: Initialize iteration-specific datasets for collecting labeled data: $\mathcal{D}'_{\text{train}} \leftarrow \emptyset$, $\mathcal{D}'_{\text{val}} \leftarrow \emptyset$
9: **while** $B_m > 0$ **do**
10: Train N concept models $\{\tilde{g}_i\}_{i=1}^{N}$ on $\mathcal{D}_{\text{train}} \cup \mathcal{D}'_{\text{train}}$, using $\mathcal{D}_{\text{val}} \cup \mathcal{D}'_{\text{val}}$ for early stopping
11: Calculate acquisition function value $\alpha(s)$ for all $s \in \mathcal{U}_m \setminus (\mathcal{D}'_{\text{train}} \cup \mathcal{D}'_{\text{val}})$
12: Choose $b_m = \min(b, B_m)$ unlabeled points from $\mathcal{U}_m$ according to $\arg\max_s \alpha(s)$
13: Query for concept labels to obtain new dataset $\mathcal{D}_m \leftarrow \{(s, c)\}^{b_m}$
14: Split $\mathcal{D}_m$ into train and validation splits and add to $\mathcal{D}'_{\text{train}}, \mathcal{D}'_{\text{val}}$
15: Decrement budget: $B_m \leftarrow B_m - b_m$
16: **end while**
17: Aggregate datasets: $\mathcal{D}_{\text{train}} \leftarrow \mathcal{D}_{\text{train}} \cup \mathcal{D}'_{\text{train}}$, $\mathcal{D}_{\text{val}} \leftarrow \mathcal{D}_{\text{val}} \cup \mathcal{D}'_{\text{val}}$
18: Continue training the concept network g on $\mathcal{D}_{\text{train}}$, using $\mathcal{D}_{\text{val}}$ for early stopping
19: Freeze g and continue training f using standard RL training to obtain π_{m+1}
20: **end for**

concept learning and behavior learning. Assume that we have access to an annotated concept dataset $\mathcal{D}_{\text{train}}$ based on the the data collected from our current policy π_m. Then, *concept learning* focuses on training the concept portion of the network g on this dataset $\mathcal{D}_{\text{train}}$ (lines 17 to 19). *Behavior learning* involves freezing g and training the behavior portion of the network f with any standard RL algorithm with its associated loss L^{RL} (line 19). Upon completion of behavior learning, we obtain an updated policy π_{m+1}, which serves to collect more unlabeled concept data to help train g in the next iteration.

Data Decorrelation In the previous section, we omitted that we do not obtain ground-truth concept labels when rolling out the current policy π_m. Instead, rollouts collect a dataset $\mathcal{U}_m$ of unlabeled states as candidates for querying for ground-truth concepts from an oracle. This setup raises the question of how to collect diverse and informative data, given that policy rollouts are time-consuming. We observe that simply including all of the states encountered during the first $|\mathcal{U}_m|$ rollout steps may mean that large chains of highly similar states would be included in the dataset, therefore decreasing the data diversity. To address this challenge, we introduce data decorrelation. By introducing τ as a hyperparameter that acts as a budget multiplier to determine the size of $\mathcal{U}_m$, such that $|\mathcal{U}_m| = \tau \cdot B_m$, and, more importantly, implementing an acceptance probability p for each state, we improve the diversity of our dataset (line 6). Randomly accepting or rejecting consecutive data points helps reduce chains of highly similar states, as the data generated by an RL policy is temporally correlated. For a Markov process, as samples become more distant in time, they gradually approach independence at a rate determined by the process's mixing time.

Disagreement-Based Active Learning Equipped with our unlabeled dataset, we are now prepared to select data points for querying for concept labels. The purpose of this stage is to make use of our limited labeling budget B_m to collect a labeled dataset $\mathcal{D}_m$ for training g. We propose to train a concept ensemble—consisting of N independent concept models—from scratch on the dataset of training points $\mathcal{D}_{\text{train}}$ that has been aggregated over all iterations (line 10). We use this ensemble to calculate the disagreement-based acquisition function $\alpha(\cdot)$, which determines whether each candidate in our unlabeled dataset $\mathcal{U}_m$ ought to receive a ground-truth concept label (line 11). This function targets samples where prediction disagreement is highest among ensemble members, as these points often represent areas of uncertainty or decision boundaries where additional labeled data would be most informative. For $\alpha(\cdot)$, we use two different formulations, depending on the concept learning task. For concept classification, we use a query-by-committee (Seung et al., 1992) approach, in which we prioritize points with a high proportion of predicted class labels that are not

the modal class prediction (also called the variation ratio (Beluch et al., 2018)):

$$\alpha(s) = 1 - \max_{c \in C} \frac{1}{N} \sum_{i=1}^{N} [\tilde{g}_i(s) = c].$$

Here, $\tilde{g}_i(s)$ is the concept prediction of the i-th model on state s. For concept regression, we directly use the estimate of variance across the concept models as a measure of disagreement:

$$\alpha(s) = \sigma^2(s) = \frac{1}{N-1} \sum_{i=1}^{N} (\tilde{g}_i(s) - \mu(s))^2,$$

where $\mu(s) = \frac{1}{N} \sum_{i=1}^{N} \tilde{g}_i(s)$ is the mean prediction of the N concept models. After querying for B_m ground-truth concept labels (line 13) using this acquisition function, we are prepared to continue training g during the concept learning stage.

Implementation details For training g (line 18), we employ two types of loss functions depending on the nature of the concepts: MSE for continuous concepts and cross-entropy loss for categorical concepts. If the problem requires mixed-type concepts, we either discretize continuous attributes, converting them into categorical forms suitable for classification, or we simply use a mixed loss. For training f (line 19), we freeze g and use the PPO algorithm to continue training f from the previous iterations. For complex environments that require many iterations, we note that the RL-related parameters may get stuck in a local region in early iterations and become hard to optimize later. To mitigate this optimization issue and ensure training efficiency, we fine-tune f for all iterations to get π'_{M+1}, and in the last iteration we additionally train a new RL network f' from scratch with π'_{M+1} as the anchoring policy to get the final π_{M+1}. The anchoring policy ensures the trained π_{M+1} has a similar distribution to the previous one so the annotated observations are still highly relevant. Specifically, we create another policy instance with the same and freezed g parameters and randomly initialized f, train it with the standard PPO loss function and an additional KL-divergence penalty between the current policy and the anchoring policy:

$$L(\theta) = L^{\text{PPO}}(\theta) + \beta \cdot \text{KL}[\pi'_{M+1}(\cdot \mid s), \pi_\theta(\cdot \mid s)].$$

4 Experiments

In our experiments, we investigate the following questions:

- **RQ 1** Under a limited concept annotation budget, does LICORICE enable both high concept accuracy and high environment reward?
- **RQ 2** How effective are VLMs as automated concept annotators when used with LICORICE?
- **RQ 3** How label efficient is LICORICE compared with other methods?
- **RQ 4** Does LICORICE support test-time concept interventions?

4.1 Experiment Setup

In each experiment, we run each algorithm 5 times, each with a random seed. All algorithms use PPO (Schulman et al., 2017; Raffin et al., 2021) with a concept bottleneck. More implementation details and hyperparameters are in Appendix A.2.

Environments We investigate these questions across four distinct environments: PixelCartPole (Yang et al., 2021), DoorKey (Chevalier-Boisvert et al., 2023), DynamicObstacles (Chevalier-Boisvert et al., 2023), and Boxing (Bellemare et al., 2013). Each environment includes a distinct challenge and features a set of interpretable concepts describing key object properties. We summarize the environment properties in Table 1, with more details in Appendix A.1. These environments are characterized by their dual representation: a complex image-based input and a symbolic concept-based representation. PixelCartPole, DoorKey, and DynamicObstacles are simpler because we can extract noiseless ground-truth concept labels from their source code. In contrast, Boxing, as implemented in OCAtari (Delfosse et al., 2023), uses reverse engineering to extract positions of important objects from the game's RAM state. This extraction process introduces a small amount of noise.

Environment	# Concepts	Concept Type	Binarized # Concepts
PixelCartPole	4	Continuous	N/A
DoorKey	12	Discrete	46
DynamicObstacles	11	Discrete	30
Boxing	8	Discrete	1480

Table 1: Summary of environments, including their associated concepts. Counts for the binarized version of the concepts provided where applicable to illustrate the problem size.

Baselines To our knowledge, there exist no previous algorithms that seek to minimize the number of concept labels for interpretable RL, so we implement three: Sequential-Q, Disagreement-Q, and Random-Q. In Sequential-Q, the agent spends all of B queries on the first B states it encounters during the initial policy rollout. In Disagreement-Q, the agent similarly spends its budget at the beginning of its learning process; however, it uses active learning with the same $\alpha(\cdot)$ as LICORICE to strategically choose the training data. In Random-Q, the agent receives B concept labels at random points in the training process using a probability to decide whether to query for a concept for each state. As a budget-unconstrained baseline, we implement CPM from previous work in multi-agent RL (Zabounidis et al., 2023) for the single-agent setting. As mentioned in Section 2, CPM jointly trains the concept bottleneck and the policy, assuming unlimited access to concept labels, meaning it also represents an upper bound on concept accuracy.

VLM Details For our VLM experiments, we use GPT-4o (gpt). During training, we query GPT-4o each time LICORICE requires a concept label. As an example, for the concept related to the position of an object, we prompt GPT-4o with instructions to report the coordinates (x y). More details about our prompts can be found in Appendix A.3.

Performance Metrics We present the reward as a function of the upper bound calculated by training PPO with ground-truth concept labels (PixelCartPole: 500, DoorKey: 0.97, DynamicObstacles: 0.91, Boxing: 86.05). Percentages (or ratios) make sense since the minimum reasonable reward is 0 in all environments.[1] We additionally report the concept error (MSE for regression; 1 - accuracy for classification). In Boxing, following the practice of OCAtari, we consider a concept prediction correct if it is within 5 pixels of the ground truth label. All the reported numbers are calculated during the testing stage, where we evaluate the models on 100 episodes.

4.2 Results

RQ1: Reward and concept accuracy under a limited annotation budget

LICORICE achieves similarly high reward and concept performance to the state-of-the-art budget-unconstrained baseline. We first seek to understand how LICORICE performs compared to the state-of-the-art in concept-based RL, CPM, which is not constrained by concept label budgets. Surprisingly, LICORICE and CPM achieve nearly 100% of the maximum reward in all environments in this unfair comparison (pink and brown bars, Figure 2). It also achieves a similar concept error rate, only seeing a small increase in error for the most complex environment, Boxing. We emphasize that CPM is given an unlimited budget, and in fact, it uses over 1M concept labels for all environments (even 10M for the complex environment Boxing), which is at least $2000\times$ the budget used by LICORICE across all environments. In contrast, LICORICE operates under a strict budget constraint, with 5000 labels for Boxing, and at most 500 concept labels for the simpler environments (PixelCartPole: 500, DoorKey: 300, DynamicObstacles: 300).

LICORICE generally largely outperforms budget-constrained baselines in terms of both reward and concept error. We now investigate how LICORICE performs compared to other algorithms with budget constraints. As a result, we study LICORICE under the same fixed concept labeling budget B as above against the budget-constrained baselines described in Section 4.1. Figure 2 shows the results, which reveal that LICORICE outperforms all baselines in terms of both reward and concept error on all but arguably the easiest environment, DynamicObstacles. In that

[1] In PixelCartPole and DoorKey, all rewards are nonnegative; in DynamicObstacles, an agent can ensure nonnegative reward by simply staying in place; in Boxing, a random policy can get around 0 reward and all of the trained policies have positive reward.

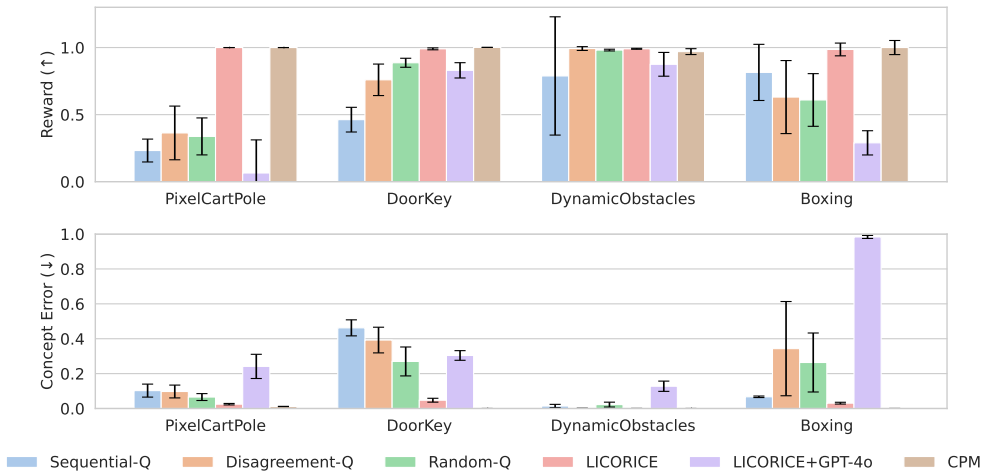

Figure 2: Reward R and concept error achieved by all methods in all environments. The first 5 methods are given a budget of $B = [500, 300, 300, 5000]$, respectively; CPM is given an unlimited budget (in practice, it uses 4M, 4M, 1M, 10M concept labels respectively). Full results with standard deviation are in Table 5, Appendix B.

environment, LICORICE performs similarly to the baselines. The main performance differences occur in PixelCartPole and Boxing, where LICORICE achieves 100% and 99% of the maximum reward, respectively, while the second-best algorithm achieves 36% and 81%, respectively. We therefore answer **RQ 1** in the affirmative: not only does LICORICE achieve the same high concept accuracy and high reward as the state-of-the-art in concept-based RL, it also performs on-par to budget-constrained baselines in one environment and outperforms them in the other three.

RQ 2: VLMs as concept annotators

We now seek to answer the question of whether VLMs can successfully provide concept labels in lieu of a human annotator within our LICORICE framework. Because using VLMs incurs costs and users requiring interpretable policies for their environments may still face budget constraints, we operate within the same budget-constrained setting as above. We present these results in Figure 2.

VLMs can serve as concept annotators for some environments. In DoorKey and DynamicObstacles, LICORICE with GPT-4o labels achieves 83% and 88% of the maximum reward, respectively. To assess the quality of VLM-generated labels, we compare the concept error rate of our trained model against GPT-4o's labeling error, both evaluated on the same rollout observations. Due to computational constraints, we apply sampling to GPT-4o evaluations. Table 2 shows that the concept error of LICORICE is comparable to that of GPT-4o when GPT-4o labeling error is not high, which suggests that LICORICE can effectively utilize these concept labels. This indicates that VLMs, particularly GPT-4o, could serve as viable concept annotators for certain environments.

	c Error	
Environment	LICORICE+GPT-4o	GPT-4o
PixelCartPole	0.25	0.24
DoorKey	0.31	0.30
DynamicObstacles	0.13	0.13
Boxing	0.98	0.82

Table 2: Concept error comparison. The concept error of LICORICE+GPT-4o matches that of GPT-4o alone.

VLMs struggle to provide accurate concepts in more complex environments. In PixelCartPole and Boxing, however, GPT-4o faces challenges in providing accurate labels. Not only does LICORICE+GPT-4o struggle to achieve even 35% of the maximum reward in either environment, it also incurs a large concept error. The challenge with PixelCartPole is the continuous nature of the concepts and the lack of necessary knowledge of physical rules and quantities in this specific environment. Boxing is particularly challenging due to the large number of possible concept values that each of the 8 discrete concepts can take on (160 or 210 possible values). We therefore answer **RQ 2** with cautious optimism: there are indeed cases in which LICORICE could be used alongside

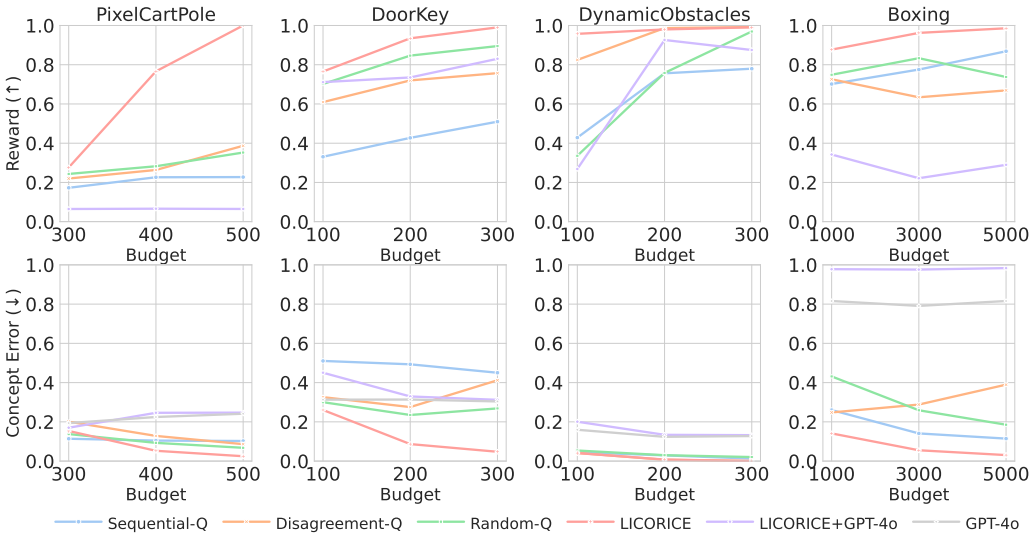

Figure 3: Reward (top) and concept error (bottom) of all algorithms for all environments across different budgets. Overall, LICORICE more efficiently makes use of the varying budgets, achieving higher reward and lower concept error.

VLMs. Generally speaking, a more human-intuitive and less complex environment is more likely to work well with VLMs by enabling clear and detailed labeling instructions in the prompt. However, in safety-critical settings, VLM capabilities will need to improve before they can be used to fully alleviate the human annotation burden.

RQ 3: Investigation of Budget Requirements for Performance

Given these results, we now investigate the minimum budget required for all environments to achieve 99% of the reward upper bound. As a result, we incrementally increase the budget for each environment starting from a reasonably small budget until LICORICE reaches 99% of the reward upper bound. In addition to the baselines, we study LICORICE under perfect (human) annotation and noisy VLM labels in Figure 3.

LICORICE more rapidly achieves high reward compared to baselines. Across all environments, LICORICE is the most query efficient. In three environments, it consistently achieves higher reward than baselines across all budget levels. In the fourth environment, LICORICE outperforms baselines at the smallest query budget, after which one Disagreement-Q achieves comparable reward. LICORICE is similarly performant with respect to concept error. Except for one budget setting for one environment, LICORICE consistently achieves lower concept error than the baselines.

For LICORICE+GPT-4o, more concept labels is not always better. In DoorKey, LICORICE+GPT-4o exhibits predictable behavior in terms of concept error: as the budget increases, the reward increases and the concept error decreases. Counterintuitively, for some environments, the concept error and reward fluctuates with more budget, likely due to the additional labeling noise introduced to the concept network. We therefore provide a more nuanced answer to **RQ 3**: LICORICE is indeed label efficient across varying budgets, but the benefit is annotator-dependent.

RQ 4: Interpretability Analysis: Test-Time Concept Interventions

A great property of concept-based networks is the ability for people to successfully intervene on the concepts to correct them. In RL, this intervention enables the inspection of how different concepts influence the immediate decisions of the agent.

LICORICE enables test-time concept intervention. We validate that our algorithm supports test-time intervention for concept-based RL. Specifically, we simulate using a noisy concept model with our trained policy, studying the impact of concept interventions on the final reward. For PixelCartPole, this means each concept adds a Gaussian noise with a standard deviation of 0.2. For the Min-

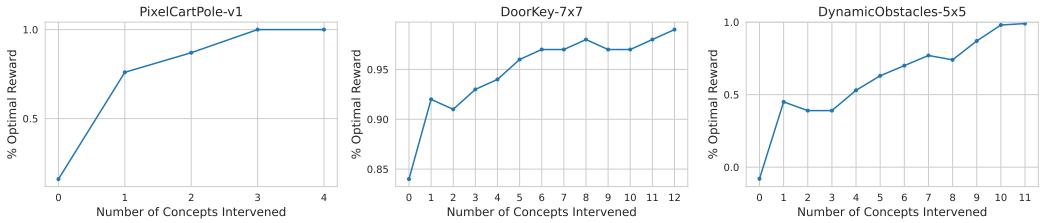

Figure 4: Concept intervention results: LICORICE enables test-time concept intervention.

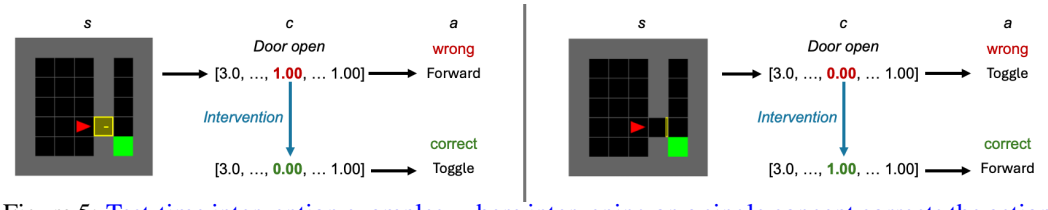

Figure 5: Test-time intervention examples, where intervening on a single concept corrects the action.

igrid environments, the label of each concept is randomly changed with probability 0.2. Following previous work (Koh et al., 2020), we first intervene individually on concepts, setting them to ground truth, and sort the concepts in descending order of reward improvement. We then sequentially intervene on the concepts following this ordering, such that any previously intervened concepts remain intervened. According to Figure 4, using a noisy concept model significantly degrades reward for all environments. However, the performance steadily increases as we intervene on more concepts, meaning LICORICE indeed supports test-time concept intervention, affirmatively answering **RQ 4**.

Concept interventions show how the policy decisions change. We now show how domain experts could interact with the model. Specifically, we simulate a counterfactual question: What if a concept value is incorrectly predicted? Figure 5 depicts two examples in DoorKey. Just intervening on the concept of the door being open or closed is sufficient to change the agent's behavior, both highlighting the importance of this specific concept and the ability of a user to intervene on the concepts to interrogate and understand agent behavior.

Additional Experiments

Ablation study: all components of LICORICE contribute positively to its performance. We now conduct ablations to confirm the effectiveness of our three main contributions: iterative training, decorrelation, and disagreement-based active learning. LICORICE-IT corresponds to LICORICE with only one iteration, LICORICE-DE corresponds to LICORICE without decorrelation, and LICORICE-AC corresponds to LICORICE without active learning (instead, it uses the entire unlabeled dataset for querying). We show the learning curves in Appendix B. All of our contributions are critical to achieving both high reward and low concept error (Table 3, bottom row corresponds to the upper bound on performance by LICORICE). However, the component that most contributes to the performance differs depending on the environment. For example, compared with LICORICE, LICORICE-IT exhibits the largest reward gap for PixelCartPole; however, LICORICE-AC yields the largest reward gap for DynamicObstacles, and LICORICE-DE yields the largest gap for DoorKey. We suspect that this difference is because the concepts in DynamicObstacles are simple enough such that one iteration is sufficient for learning, meaning that the largest gains can be made by using active learning. In contrast, PixelCartPole requires further policy refinement to better estimate on-distribution concept values, so the largest gains can be made by leveraging multiple iterations.

LICORICE is robust to various hyperparameter values. LICORICE involves a few key hyperparameters, described in Section 3 and summarized here for convenience. In data decorrelation, τ governs the size of the unlabeled dataset collected by the current policy and p controls the rate at which we accept data points. During the disagreement-based active learning, we set a committee size N for the number of concept models in the ensemble. We study the effect of the values of these hyperparameters on DynamicObstacles, finding that LICORICE achieves $96-99\%$ of the maximum reward across 2 values for N, 3 values for p, and 3 values for τ (Appendix B.5).

Algorithm	PixelCartPole		DoorKey		DynamicObstacles		Boxing	
	$R\uparrow$	c MSE $\downarrow$	$R\uparrow$	c Error $\downarrow$	$R\uparrow$	c Error $\downarrow$	$R\uparrow$	c Error $\downarrow$
LICORICE-IT	0.53	0.08	0.99	0.09	1.00	0.00	0.94	0.19
LICORICE-DE	0.97	0.03	0.78	0.20	0.98	0.00	0.94	0.04
LICORICE-AC	0.91	0.02	0.82	0.16	0.58	0.00	0.91	0.05
LICORICE	1.00	0.02	0.99	0.05	0.99	0.00	0.99	0.03

Table 3: Ablation study results for LICORICE. All components generally help achieve better reward and lower concept error. Full results with standard deviation are in Table 8, Appendix B.

5 Related Work

Interpretable RL Interpretable RL has gained significant attention in recent years (Glanois et al., 2024). One prominent approach uses rule-based methods—such as decision trees (Silva et al., 2020; Topin et al., 2021), logic (Delfosse et al., 2024), and programs (Verma et al., 2018; Penkov & Ramamoorthy, 2019)—to represent policies. These works either assume that the state is already interpretable or that the policy is pre-specified. Unlike prior work, our method involves *learning* the interpretable representation (through concept training) for policy learning.

Concept Learning for RL Inspired by successes in the supervised setting (Collins et al., 2023; Sheth & Ebrahimi Kahou, 2023; Zarlenga et al., 2023), concept models have recently been used in RL. In addition to interpretability, concept learning offers a powerful path to compositional generalization by helping RL agents decompose complex tasks into meaningful, reusable components that can be flexibly recombined (Mao et al., 2022; Wang et al., 2023). One approach (Das et al., 2023) learns a joint embedding model between state-action pairs and concept-based explanations to expedite learning via reward shaping. Unlike LICORICE, their policy is not a strict function of the concepts, allowing our techniques to be combined to provide both concept-based explanations and a concept-based interpretable policy. Another example, CPM (Zabounidis et al., 2023), is a multi-agent RL concept architecture that focuses on trade-offs between interpretability and accuracy. They assume that concept labels are available continuously during training. As we have shown, this approach uses over 1M concept labels, whereas our approach reduces the need for continuous human intervention, requiring only $200 - 2000\times$ fewer concept labels to achieve similar performance in single-agent environments.

Learning Concepts with Human Feedback Previous research has explored leveraging human concept labels, but not for RL and without focusing on reducing the labeling burden. For instance, Lage & Doshi-Velez (2020) has users label additional information about the relevance of certain feature dimensions to the concept label to facilitate concept learning for high-dimensional data. In a different vein, Chauhan et al. (2023) develop an intervention policy for test-time selection of concepts requiring labels with the goal of improving the final prediction. Although these approaches offer valuable insights, they do not directly tackle the issue of reducing the overall labeling burden during training and could be used alongside our method.

6 Discussion and Conclusion

In this work, we proposed LICORICE, a novel RL algorithm that addresses the critical issue of model interpretability while minimizing the reliance on continuous human annotation. We introduced a training scheme that enables RL algorithms to learn concepts more efficiently from little to no labeled concept data. Our approach interleaves concept learning and RL training, uses an ensemble-based active learning technique to select informative data points for labeling, and uses a simple sampling strategy to better decorrelate the concept data. We demonstrated how this approach reduces manual labeling effort. Finally, we conducted initial experiments to demonstrate how we can leverage powerful VLMs to infer concepts from raw visual inputs without explicit labels in some environments. There are broader societal impacts of our work that must be considered. These include both the impacts of using VLMs in real-world applications, as well as considerations around interpretability more generally. For a more detailed discussion of limitations and future work, please refer to Appendix C.

REFERENCES

Hello gpt-4o. https://openai.com/index/hello-gpt-4o/.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

William H. Beluch, Tim Genewein, A. Nürnberger, and Jan M. Köhler. The power of ensembles for active learning in image classification. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9368–9377, 2018. URL https://api.semanticscholar.org/CorpusID:52838058.

Kushal Chauhan, Rishabh Tiwari, Jan Freyberg, Pradeep Shenoy, and Krishnamurthy Dvijotham. Interactive concept bottleneck models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 5948–5955, 2023.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

Katherine Maeve Collins, Matthew Barker, Mateo Espinosa Zarlenga, Naveen Raman, Umang Bhatt, Mateja Jamnik, Ilia Sucholutsky, Adrian Weller, and Krishnamurthy Dvijotham. Human uncertainty in concept-based ai systems. In *Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 869–889, 2023.

Devleena Das, Sonia Chernova, and Been Kim. State2explanation: Concept-based explanations to benefit agent learning and user understanding. *Advances in Neural Information Processing Systems*, 36, 2023.

Quentin Delfosse, Jannis Blüml, Bjarne Gregori, Sebastian Sztwiertnia, and Kristian Kersting. Ocatari: Object-centric atari 2600 reinforcement learning environments. 2023.

Quentin Delfosse, Hikaru Shindo, Devendra Dhami, and Kristian Kersting. Interpretable and explainable logical policies via neurally guided symbolic abstraction. *Advances in Neural Information Processing Systems*, 36, 2024.

Mateo Espinosa Zarlenga, Pietro Barbiero, Gabriele Ciravegna, Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, Zohreh Shams, Frederic Precioso, Stefano Melacci, Adrian Weller, et al. Concept embedding models: Beyond the accuracy-explainability trade-off. *Advances in Neural Information Processing Systems*, 35:21400–21413, 2022.

Claire Glanois, Paul Weng, Matthieu Zimmer, Dong Li, Tianpei Yang, Jianye Hao, and Wulong Liu. A survey on interpretable reinforcement learning. *Machine Learning*, pp. 1–44, 2024.

Niko Grupen, Natasha Jaques, Been Kim, and Shayegan Omidshafiei. Concept-based understanding of emergent multi-agent behavior. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

Harmanpreet Kaur, Harsha Nori, Samuel Jenkins, Rich Caruana, Hanna Wallach, and Jennifer Wortman Vaughan. Interpreting interpretability: understanding data scientists' use of interpretability tools for machine learning. In *Proceedings of the 2020 CHI conference on human factors in computing systems*, pp. 1–14, 2020.

Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *International Conference on Machine Learning*, pp. 5338–5348. PMLR, 2020.

Isaac Lage and Finale Doshi-Velez. Learning interpretable concept-based models with human feedback. *International Conference on Machine Learning: Workshop on Human Interpretability in Machine Learning*, 2020.

Xiao-Yang Liu, Ziyi Xia, Jingyang Rui, Jiechao Gao, Hongyang Yang, Ming Zhu, Christina Wang, Zhaoran Wang, and Jian Guo. Finrl-meta: Market environments and benchmarks for data-driven financial reinforcement learning. *Advances in Neural Information Processing Systems*, 35:1835–1849, 2022.

Jiayuan Mao, Tomás Lozano-Pérez, Josh Tenenbaum, and Leslie Kaelbling. Pdsketch: Integrated domain programming, learning, and planning. *Advances in Neural Information Processing Systems*, 35:36972–36984, 2022.

Catherine C Marshall and Frank M Shipman. Experiences surveying the crowd: Reflections on methods, participation, and reliability. In *Proceedings of the 5th Annual ACM Web Science Conference*, pp. 234–243, 2013.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

Hussein Mozannar, Hunter Lang, Dennis Wei, Prasanna Sattigeri, Subhro Das, and David Sontag. Who should predict? exact algorithms for learning to defer to humans. In *AISTATS*, 2023.

Tuomas Oikarinen, Subhro Das, Lam M Nguyen, and Tsui-Wei Weng. Label-free concept bottleneck models. *arXiv preprint arXiv:2304.06129*, 2023.

Bruno A Olshausen and David J Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

Svetlin Penkov and Subramanian Ramamoorthy. Learning programmatically structured representations with perceptor gradients. In *Proceedings of the International Conference on Learning Representations*, 2019.

Eleonora Poeta, Gabriele Ciravegna, Eliana Pastor, Tania Cerquitelli, and Elena Baralis. Concept-based explainable artificial intelligence: A survey. *arXiv preprint arXiv:2312.12936*, 2023.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

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

H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 287–294, 1992.

Ivaxi Sheth and Samira Ebrahimi Kahou. Auxiliary losses for learning generalizable concept-based models. *Advances in Neural Information Processing Systems*, 36, 2023.

Andrew Silva, Matthew Gombolay, Taylor Killian, Ivan Jimenez, and Sung-Hyun Son. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In *International conference on artificial intelligence and statistics*, pp. 1855–1865. PMLR, 2020.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Alaa Tharwat and Wolfram Schenck. A survey on active learning: state-of-the-art, practical challenges and research directions. *Mathematics*, 11(4):820, 2023.

Nicholay Topin, Stephanie Milani, Fei Fang, and Manuela Veloso. Iterative bounding mdps: Learning interpretable policies via non-interpretable methods. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 9923–9931, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 10, 2017.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pp. 5045–5054. PMLR, 2018.

Renhao Wang, Jiayuan Mao, Joy Hsu, Hang Zhao, Jiajun Wu, and Yang Gao. Programmatically grounded, compositionally generalizable robotic manipulation. *International Conference on Learning Representations*, 2023.

Chao-Han Huck Yang, I Danny, Te Hung, Yi Ouyang, and Pin-Yu Chen. Causal inference q-network: Toward resilient reinforcement learning. In *Self-Supervision for Reinforcement Learning Workshop-ICLR 2021*, 2021.

Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, 55(1):1–36, 2021.

Renos Zabounidis, Joseph Campbell, Simon Stepputtis, Dana Hughes, and Katia P Sycara. Concept learning for interpretable multi-agent reinforcement learning. In *Conference on Robot Learning*, pp. 1828–1837. PMLR, 2023.

Mateo Espinosa Zarlenga, Katherine M Collins, Krishnamurthy Dvijotham, Adrian Weller, Zohreh Shams, and Mateja Jamnik. Learning to receive help: Intervention-aware concept embedding models. *Advances in Neural Information Processing Systems*, 2023.

Environment	Concept Name	Type	Value Ranges	GPT-4o Error
PixelCartPole	Cart Position	Continuous	$(-2.4, 2.4)$	0.01 ± 0.00
	Cart Velocity	Continuous	$\mathbb{R}$	0.31 ± 0.12
	Pole Angle	Continuous	$(-.2095, .2095)$	0.01 ± 0.00
	Pole Angular Velocity	Continuous	$\mathbb{R}$	0.63 ± 0.14
DoorKey	Agent Position x	Discrete	5	0.36 ± 0.08
	Agent Position y	Discrete	5	0.41 ± 0.12
	Agent Direction	Discrete	4	0.28 ± 0.06
	Key Position x	Discrete	6	0.20 ± 0.03
	Key Position y	Discrete	6	0.32 ± 0.11
	Door Position x	Discrete	5	0.37 ± 0.07
	Door Position y	Discrete	5	0.32 ± 0.04
	Door Open	Discrete	2	0.38 ± 0.09
	Direction Movable Right	Discrete	2	0.30 ± 0.03
	Direction Movable Down	Discrete	2	0.19 ± 0.06
	Direction Movable Left	Discrete	2	0.33 ± 0.06
	Direction Movable Up	Discrete	2	0.20 ± 0.08
DynamicObstacles	Agent Position x	Discrete	3	0.03 ± 0.06
	Agent Position y	Discrete	3	0.04 ± 0.06
	Agent Direction	Discrete	4	0.20 ± 0.05
	Obstacle 1 Position x	Discrete	3	0.08 ± 0.02
	Obstacle 1 Position y	Discrete	3	0.19 ± 0.04
	Obstacle 2 Position x	Discrete	3	0.22 ± 0.09
	Obstacle 2 Position y	Discrete	3	0.12 ± 0.02
	Direction Movable Right	Discrete	2	0.19 ± 0.06
	Direction Movable Down	Discrete	2	0.14 ± 0.05
	Direction Movable Left	Discrete	2	0.13 ± 0.02
	Direction Movable Up	Discrete	2	0.07 ± 0.08
Boxing	Agent Position x at Frame 1	Discrete	160	0.81 ± 0.14
	Agent Position y at Frame 1	Discrete	210	0.93 ± 0.05
	Enemy Position x at Frame 1	Discrete	160	0.78 ± 0.13
	Enemy Position y at Frame 1	Discrete	210	0.90 ± 0.07
	Agent Position x at Frame 2	Discrete	160	0.67 ± 0.06
	Agent Position y at Frame 2	Discrete	210	0.88 ± 0.04
	Enemy Position x at Frame 2	Discrete	160	0.71 ± 0.05
	Enemy Position y at Frame 2	Discrete	210	0.83 ± 0.08

Table 4: Concepts and their possible values for all environments. For discrete concepts, we report the number of categories. We also provide the mean GPT-4o labeling error for each single concept, averaged across 5 seeds and corresponding to the same evaluation protocols as the ones with the largest budgets in Table 7. We use MSELoss for continuous values and 1 - accuracy for discrete ones, and in Boxing, we regard one prediction correct if it is within distance 5 to the ground truth. The $\pm$ [value] part shows the standard deviation. The average errors over concepts are 0.24 ± 0.06, 0.30 ± 0.02, 0.13 ± 0.03, and 0.82 ± 0.06 respectively for the four environments.

A Experimental Result Reproducibility

In this section, we provided detailed descriptions of our concept definitions, prompts, and other experimental details toward the goal of reproducibility.

A.1 Concepts Definitions

Table 4 provides more details regarding the concepts used in each environment, categorizing them by their names, types, and value ranges. For the PixelCartPole environment, all concepts such as Cart Position, Cart Velocity, Pole Angle, and Pole Angular Velocity are continuous. In contrast, the DoorKey environment features discrete concepts like Agent Position (x and y), Key Position (x and y), and Door Open status, each with specific value ranges. Similarly, the DynamicObstacles

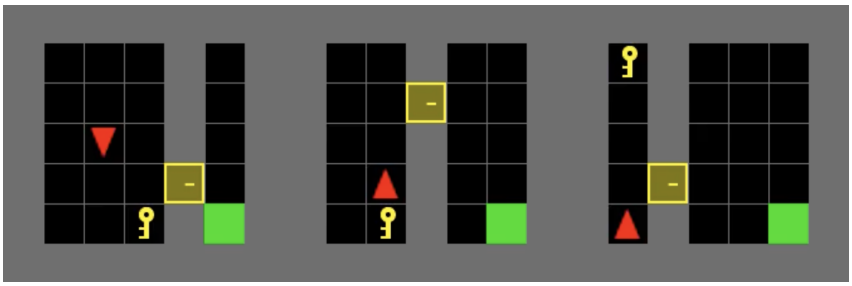

Figure 6: Example start configurations of the DoorKey environment. This shows that concepts must be general enough to apply to many different configurations. As a result, the agent cannot memorize the exact position of the door and key.

environment lists discrete concepts, including Agent and Obstacle positions, with corresponding value ranges.

Precise concept definitions are crucial for enabling correct agent behavior. We visualize the start configurations for DoorKey in Figure 6 to illustrate the importance of well-defined concepts in reinforcement learning. DoorKey includes a variety of initial states, each with different positions of the agent, key, and door. This diversity highlights a fundamental challenge in concept-based RL: how to define concepts that are both specific enough to be meaningful and general enough to apply across all possible scenarios. An agent trained with poorly defined concepts might perform well in some configurations but fail to generalize to others, leading to suboptimal performance in new or unseen environments. As a result, concept engineering is its own separate but important problem.

A.2 LICORICE Details

In this section, we provide additional implementation details for LICORICE. The model architecture has been mostly described in the main text and we state all additional details here. The number of neurons in the concept layer is exactly the number of concepts. For continuous concept values, we directly use a linear layer to map from features to concept values. For discrete concept values, since different concepts have different numbers of categories, we create one linear classification head for each single concept, and to predict the final action, we calculate the class with the largest predicted probability for each concept.

Architecture The network begins with a CNN-based feature extractor $f_\theta : \mathcal{X} \to \mathbb{R}^d$, comprising three convolutional layers that map input state $x \in \mathcal{X}$ to a d-dimensional feature vector $h = f_\theta(x)$. We then introduce a linear layer $g_\phi : \mathbb{R}^d \to \mathbb{R}^k$ for concept prediction (g), mapping extracted features to k concept values. For regression tasks, each of the k predicted concept is represented as a real value; for classification tasks, each concept maps to one categorical value derived from a classification head. The action prediction component (f) is implemented as an MLP with two fully connected layers, each containing 64 neurons with Tanh activation, taking only c as input to produce action logits $a = \text{MLP}(c)$. Notably, we share the CNN feature extractor f_θ between policy and value functions, a decision informed by improved performance observed in preliminary experiments.

Value-Based Methods If we were to use a value-based method as the RL backbone, we would need to make the following changes. First, we would need to modify $V(s, a)$ or $Q(s, a)$ to include a concept bottleneck, such that $Q(s, a) = f(g(s))$. Then, we can conduct interleaved training in a similar way to LICORICE.

Feature Extractor If we use the actor-critic paradigm, we propose to share a feature extractor between the policy and value networks, shown in Figure 7. Intuitively, this choice can offer several advantages compared with using image or predicted concepts as input for both networks. Sharing a feature extractor enables both networks

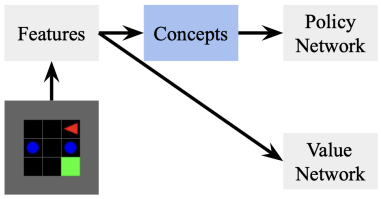

Figure 7: Architecture of our concept-bottleneck actor-critic method.

to benefit from a common, rich representation of the input data, reducing the number of parameters to be trained. More importantly, it balances the updates of the policy and value networks. In experiments, we observed that directly using the raw image as input for both networks complicated policy learning. Conversely, relying solely on predicted concepts for the value network may limit its accuracy in value estimation, particularly if the concepts do not capture all the nuances relevant to the value predictions.

A.3 VLM Details

We detail our prompts for each environment here.

PixelCartPole

> **Prompt:** Here are the past 4 rendered frames from the CartPole environment. Please use these images to estimate the following values in the latest frame (the last one):
>
> - Cart Position, within the range (-2.4, 2.4)
> - Cart Velocity
> - Pole Angle, within the range (-0.2095, 0.2095)
> - Pole Angular Velocity
>
> Additionally, please note that the last action taken was [last action].
>
> Please carefully determine the following values and give concise answers one by one. Make sure to return an estimated value for each parameter, even if the task may look challenging.
>
> Follow the reporting format:
>
> - Cart Position: estimated_value
> - Cart Velocity: estimated_value
> - Pole Angle: estimated_value
> - Pole Angular Velocity: estimated_value

DoorKey

Prompt: Here is an image of a 4x4 grid composed of black cells, with each cell either empty or containing an object. Each cell is defined by an integer-valued coordinate system starting at (1, 1) for the top-left cell. The coordinates increase rightward along the x-axis and downward along the y-axis. Within this grid, there is a red isosceles triangle representing the agent, a yellow cell representing the door (which may visually disappear if the door is open), a yellow key icon representing the key (which may disappear), and one green square representing the goal. Carefully analyze the grid and report on the following attributes, focusing only on the black cells as the gray cells are excluded from the active black area.

Detailed Instructions:

1. Agent Position: Identify and report the coordinates (x, y) of the red triangle (agent). Ensure the accuracy by double-checking the agent's exact location within the grid.
2. Agent Direction: Specify the direction the red triangle is facing, which is the orientation of the vertex (pointy corner) of the isosceles triangle. Choose from 'right', 'down', 'left', or 'up'. Clarify that this direction is independent of movement options.
3. Key Position: Provide the coordinates (x, y) where the key is located. If the key is absent, report as (0, 0). Verify visually that the key is present or not before reporting.
4. Door Position:
   - Position: Determine and report the coordinates (x, y) of the door.
   - Status: Assess whether the door is open or closed (closed means the door is visible as a whole yellow cell, while open means the door disappears visually). Report as 'true' for open and 'false' for closed. Double-check the door's appearance to confirm if it is open or closed.
5. Direction Movable: Evaluate and report whether the agent can move one cell in each specified direction, namely, the neighboring cell in that direction is active and empty (not key, closed door, or grey inactive cell):
   - Right (x + 1): Check the cell to the right.
   - Down (y + 1): Check the cell below.
   - Left (x - 1): Check the cell to the left.
   - Up (y - 1): Check the cell above.

   Each direction's feasibility should be reported as 'true' if clear and within the grid, and 'false' otherwise.

Reporting Format: Carefully report each piece of information sequentially, following the format 'name: answer'. Ensure each response is precise and reflects careful verification of the grid details as viewed.

DynamicObstacles

Prompt: Here is an image of a 3x3 grid composed of black cells, with each cell either empty or containing an object. Each cell is defined by an integer-valued coordinate system starting at (1, 1) for the top-left cell. The coordinates increase rightward along the x-axis and downward along the y-axis. Within this grid, there is a red isosceles triangle representing the agent, two blue balls representing obstacles, and one green square representing the goal. Please carefully determine the following values and give concise answers one by one:

1. Agent Position: Identify and report the coordinates (x, y) of the red triangle (agent). Ensure the accuracy by double-checking the agent's exact location within the grid.
2. Agent Direction: Specify the direction the red triangle is facing, which is the orientation of the vertex (pointy corner) of the isosceles triangle. Choose from 'right', 'down', 'left', or 'up'. Clarify that this direction is independent of movement options.
3. Obstacle Position: Identify and report the coordinates of the two obstacles in ascending order. Compare the coordinates by their x-values first. If the x-values are equal, compare by their y-values.
   (a) First Obstacle: Provide the coordinates (x, y) of the first blue ball.
   (b) Second Obstacle: Provide the coordinates (x, y) of the second blue ball.
4. Direction Movable: Evaluate and report whether the agent can move one cell in each specified direction, namely, the neighboring cell in that direction is active and empty (not obstacle or out of bounds):
   - Right (x + 1): Check the cell to the right.
   - Down (y + 1): Check the cell below.
   - Left (x - 1): Check the cell to the left.
   - Up (y - 1): Check the cell above.

   Each direction's feasibility should be reported as 'true' if clear and within the grid, and 'false' otherwise.

Reporting Format: Carefully report each piece of information sequentially, following the format 'name: answer'. Ensure each response is precise and reflects careful verification of the grid details as viewed.

Boxing

> **Prompt:** Here are two consecutive rendered frames from the Atari Boxing environment. The game screen is 160x210 pixels, with (0, 0) at the top-left corner. The x-coordinate increases rightward, and the y-coordinate increases downward. For each frame, estimate the following values as integers:
>
> - The white player's x and y coordinates
> - The black player's x and y coordinates
>
> Please carefully determine the following values and give concise answers one by one. Make sure to return an estimated value for each one, even if the task may look challenging.
>
> Follow the reporting format:
>
> - frame 1 white x: estimated_value
> - frame 1 white y: estimated_value
> - frame 1 black x: estimated_value
> - frame 1 black y: estimated_value
> - frame 2 white x: estimated_value
> - frame 2 white y: estimated_value
> - frame 2 black x: estimated_value
> - frame 2 black y: estimated_value

A.4 Experimental Details

For the PPO hyperparameters, we set $4 \cdot 10^6$ total timesteps for PixelCartPole and DoorKey, 10^6 for DynamicObstacles, and 10^7 for Boxing. For PixelCartPole, DoorKey, and DynamicObstacles, we use 8 vectorized environments, horizon $T = 4096$, 10 epochs for training, batch size of 512, learning rate $3 \cdot 10^{-4}$, entropy coefficient 0.01, and value function coefficient 0.5. For Boxing, we use 8 vectorized environments, horizon $T = 1024$, 4 epochs for training, batch size of 256, learning rate $3 \cdot 10^{-4}$, entropy coefficient 0.01, and value function coefficient 0.5. For all other hyperparameters, we use the default values from Stable Baselines 3 Raffin et al. (2021).

For the concept training, we set 100 epochs with Adam optimizer with the learning rate linearly decaying from $3 \cdot 10^{-4}$ to 0 for each iteration in PixelCartPole and Boxing. In DoorKey and DynamicObstacles, we use the same optimizer and initial learning rate, yet set 50 epochs instead and set early stopping with threshold linearly increasing from 10 to 20, to incentivize the concept network not to overfit in earlier iterations. The batch size is 32.

We model concept learning for PixelCartPole as a regression problem (minimizing mean squared error). We model concept learning for DoorKey, DynamicObstacles, and Boxing as classification problems.

For LICORICE, we set the sample acceptance rate $p = 0.05$, the ratio for active learning $\tau = 10$, batch size to query labels in the active learning module $b = 20$, and the number of ensemble models $N = 5$ for the first three environments. For the complex environment Boxing, we choose the sample acceptance rate $p = 0.1$, the ratio for active learning $\tau = 4$, batch size to query labels in the active learning module $b = B_m/5$ and number of ensemble models $N = 5$ to make a balance between performance and speed. The default number of iterations chosen in our algorithm is $M = 4$ for PixelCartPole, $M = 2$ for DoorKey, DynamicObstacles, and $M = 10$ for Boxing. The Random-Q baseline uses the sample acceptance rate of $p = 0.1$.

In the complex environment Boxing, due to many iterations, we use both the KL-divergence penalty and PPO loss at the end of the algorithm to improve optimization with $\beta = 0.01$, as mentioned in Section 3.

Computational resources. We use NVIDIA A6000 and NVIDIA RTX 6000 Ada Generation. Each of our training program uses less than 2GB GPU memory. For Boxing, each run takes less than

12 hours to finish. For PixelCartPole and DoorKey, each run takes less than 9 hours to finish. For DynamicObstacles, each run takes less than 2 hours to finish.

For all experiments, we consistently use 5 seeds $[123, 456, 789, 1011, 1213]$ to train the models. We then evaluate on the environment with seed 42 with 100 episodes and take the average of the reward across the episodes and the concept error across the observations within each episode.

	Algorithm	$R \uparrow$	c Error $\downarrow$
PixelCartPole	Sequential-Q	0.23 ± 0.09	0.10 ± 0.04
	Disagreement-Q	0.36 ± 0.20	0.10 ± 0.04
	Random-Q	0.34 ± 0.14	0.07 ± 0.02
	LICORICE	1.00 ± 0.00	0.02 ± 0.00
	LICORICE+GPT-4o	0.06 ± 0.01	0.25 ± 0.08
	CPM	1.00 ± 0.00	0.01 ± 0.00
DoorKey	Sequential-Q	0.46 ± 0.09	0.46 ± 0.05
	Disagreement-Q	0.76 ± 0.12	0.39 ± 0.07
	Random-Q	0.89 ± 0.03	0.27 ± 0.08
	LICORICE	0.99 ± 0.01	0.05 ± 0.01
	LICORICE+GPT-4o	0.83 ± 0.06	0.31 ± 0.01
	CPM	1.00 ± 0.00	0.00 ± 0.00
DynamicObstacles	Sequential-Q	0.79 ± 0.44	0.01 ± 0.01
	Disagreement-Q	0.99 ± 0.01	0.00 ± 0.00
	Random-Q	0.98 ± 0.01	0.02 ± 0.01
	LICORICE	0.99 ± 0.00	0.00 ± 0.00
	LICORICE+GPT-4o	0.88 ± 0.09	0.13 ± 0.03
	CPM	0.97 ± 0.02	0.00 ± 0.00
Boxing	Sequential-Q	0.81 ± 0.21	0.07 ± 0.00
	Disagreement-Q	0.63 ± 0.27	0.34 ± 0.27
	Random-Q	0.61 ± 0.20	0.26 ± 0.17
	LICORICE	0.99 ± 0.05	0.03 ± 0.01
	LICORICE+GPT-4o	0.29 ± 0.09	0.98 ± 0.01
	CPM	1.00 ± 0.05	0.00 ± 0.00

Table 5: Evaluation of the reward R and concept error achieved by all methods in all environments. This is a extended table from Figure 2. The reward is reported as the fraction of the reward upper bound. For PixelCartPole, the c error is the MSE. For the other 3 environments, the c error is 1 - accuracy. The first five algorithms are given a budget of $B = [500, 300, 300, 5000]$ for each environment, from top to bottom; CPM is given an unlimited budget (in practice, it uses 4M, 4M, 1M, 10M concept labels respectively). The $\pm$ [value] part shows the standard deviation.

B Additional Results

B.1 Balancing Concept Performance and Environment Reward

In Table 5, we present all of the numerical results from Figure 2, including standard deviation. Our method enjoys low variance across all environments in terms of both concept error and reward.

B.2 Budget Allocation Effectiveness

In Table 6, we present an extension of the results in Figure 3, including the standard deviation. As expected, as the budget increases, the standard deviation for both the reward and concept error tends to decrease. The one exception is the reward for PixelCartPole. Interestingly, the standard deviation is highest for $B = 400$. We suspect this is because the concept errors may be more critical here, leading to higher variance in the reward performance.

B.3 Integration with Vision-Language Models

In Table 7, we present an extension of the results in Figure 3 in the main paper, including the standard deviation. Interestingly, the standard deviation for the reward obtained by using LICORICE with GPT-4o as the annotator does not always follow the same trend as shown in Table 6 (when we assume access to a more accurate human annotator). Instead, the standard deviation is relatively consistent for PixelCartPole, regardless of the budget. It steadily decreases for DoorKey, as expected. However, in DynamicObstacles, we see an increase when $B = 300$. We are not sure of the cause of this.

	B	M	$R\uparrow$	c Error $\downarrow$
PixelCartPole	300	1	0.28 ± 0.10	0.15 ± 0.07
	400	3	0.77 ± 0.22	0.05 ± 0.01
	500	4	1.00 ± 0.00	0.02 ± 0.00
DoorKey	100	1	0.77 ± 0.04	0.26 ± 0.04
	200	2	0.93 ± 0.02	0.09 ± 0.01
	300	2	0.99 ± 0.01	0.05 ± 0.01
DynamicObstacles	100	1	0.96 ± 0.02	0.04 ± 0.01
	200	1	0.98 ± 0.01	0.01 ± 0.00
	300	2	0.99 ± 0.00	0.00 ± 0.00
Boxing	1000	5	0.88 ± 0.15	0.14 ± 0.05
	3000	5	0.96 ± 0.06	0.06 ± 0.01
	5000	10	0.99 ± 0.05	0.03 ± 0.01

Table 6: Performance of LICORICE on all environments for varying budgets. We select M to optimize R - c error without additional weighting since they are observed to have the same magnitude. The reward is reported as the fraction of the reward upper bound. For PixelCartPole, c error is MSE; for the other environments, c error is 1 - accuracy. The $\pm$ [value] part shows the standard deviation. This shows a more complete version of the results in Figure 3.

Environment	B	M	$R\uparrow$	LICORICE+GPT-4o c Error $\downarrow$	GPT-4o c Error
PixelCartPole	300	1	0.06 ± 0.02	0.17 ± 0.19	0.19 ± 0.25
	400	2	0.07 ± 0.02	0.25 ± 0.15	0.22 ± 0.15
	500	2	0.06 ± 0.01	0.25 ± 0.08	0.24 ± 0.07
DoorKey	100	1	0.71 ± 0.04	0.45 ± 0.04	0.31 ± 0.03
	200	2	0.74 ± 0.04	0.33 ± 0.01	0.31 ± 0.03
	300	2	0.83 ± 0.06	0.31 ± 0.01	0.30 ± 0.03
DynamicObstacles	100	1	0.27 ± 0.37	0.20 ± 0.03	0.16 ± 0.03
	200	1	0.93 ± 0.05	0.13 ± 0.03	0.12 ± 0.05
	300	1	0.88 ± 0.09	0.13 ± 0.03	0.13 ± 0.03
Boxing	1000	5	0.34 ± 0.09	0.98 ± 0.01	0.82 ± 0.05
	3000	5	0.22 ± 0.01	0.98 ± 0.01	0.79 ± 0.05
	5000	5	0.29 ± 0.09	0.98 ± 0.01	0.82 ± 0.06

Table 7: Performance of LICORICE with GPT-4o integrated into the loop for all environments across different budgets, along with the concept labeling error of GPT-4o as a reference. We select M to optimize R - c error without additional weighting since they are observed to have the same magnitude. GPT-4o c error is evaluated on a random sample of 50 observations from the same rollout set used for LICORICE+GPT-4o, due to API cost constraints. This shows a more complete version of the results in Figure 3.

Perhaps at this point the algorithm begins overfitting to the errors in the labels from GPT-4o (the concept error rate is the same for 200 and 300 labels). Further investigation is required to understand the underlying factors contributing to this anomaly.

Different tasks and concepts have various difficulties for GPT-4o. Table 4 list detailed concept errors for concepts in all environments. In PixelCartPole, cart position and pole angle have smaller errors, while velocities require understanding multiple frames and thus are harder to predict accurately. For DoorKey and DynamicObstacles, different concepts have slightly varying concept errors, indicating visual tasks have different difficulties for GPT-4o. Agent direction has as high as around 0.3 prediction error for both DoorKey and DynamicObstacles. Direction movable is also hard for DoorKey, with a high concept error even if it is a binary concept. We posit it requires the correct understanding of more than one particular object to ensure correctness. The concept accuracies in

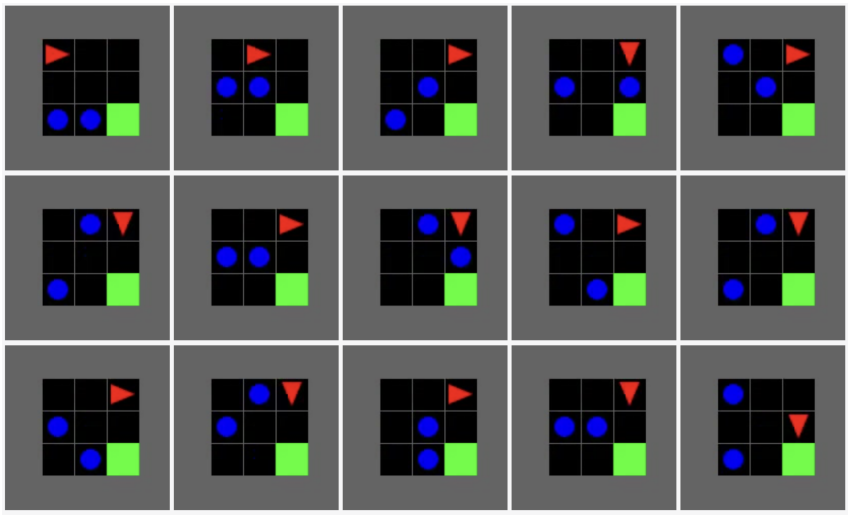

Figure 8: LICORICE+GPT-4o waits until the coast is clear to move to the goal. It appears to make a small mistake in the bottom left, requiring it to wait slightly longer than necessary to navigate to the goal.

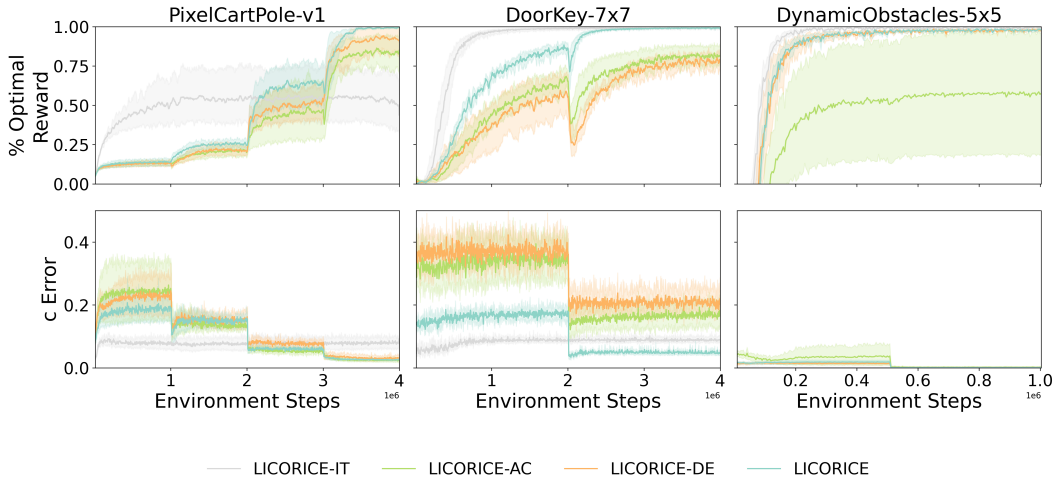

Figure 9: Learning curves for ablations in the first three environments. Shaded region shows 95% CI, calculated using 1000 bootstrap samples.

DoorKey are generally higher than DynamicObstacles, suggesting GPT-4o more struggles with a larger grid.

A GPT-4o-trained RL agent recovers from a mistake. In Figure 8, we show an example of a GPT-4o-trained RL agent on DynamicObstacles, in which the agent appears to wait until it is safe to move towards the goal: the green square. The image sequence shows the agent (red triangle) starting from its initial position and moving to the right. It then is cornered by an obstacle (blue circle), then both obstacles. In the bottom left corner frame, it appears to make a mistake by turning to the right, meaning it missed a window to escape. Finally, it moves to the goal when the path is clear in the second-to-last and last frames (bottom right). This behavior highlights that the agent may still learn reasonable behavior even if the concept labels may be incorrect (causing it to make a mistake, as in the bottom left frame).

	Algorithm	$R \uparrow$	c Error $\downarrow$
PixelCartPole	LICORICE-IT	0.53 ± 0.26	0.08 ± 0.03
	LICORICE-DE	0.97 ± 0.05	0.03 ± 0.01
	LICORICE-AC	0.91 ± 0.10	0.02 ± 0.01
	LICORICE	1.00 ± 0.00	0.02 ± 0.00
DoorKey	LICORICE-IT	0.99 ± 0.01	0.09 ± 0.02
	LICORICE-DE	0.78 ± 0.08	0.20 ± 0.05
	LICORICE-AC	0.82 ± 0.10	0.16 ± 0.06
	LICORICE	0.99 ± 0.01	0.05 ± 0.01
DynamicObstacles	LICORICE-IT	1.00 ± 0.00	0.00 ± 0.00
	LICORICE-DE	0.98 ± 0.01	0.00 ± 0.00
	LICORICE-AC	0.58 ± 0.53	0.00 ± 0.00
	LICORICE	0.99 ± 0.00	0.00 ± 0.00
Boxing	LICORICE-IT	0.94 ± 0.08	0.19 ± 0.13
	LICORICE-DE	0.94 ± 0.02	0.04 ± 0.01
	LICORICE-AC	0.91 ± 0.08	0.05 ± 0.02
	LICORICE	0.99 ± 0.05	0.03 ± 0.01

Table 8: Ablation study results for LICORICE in all environments, where we delete every single component to compare with LICORICE. This shows a more complete version of the results in Table 3.

B.4 Ablation

In Table 8, we present an extension of the results in Table 3, including standard deviation. Figure 9 shows all learning curves for ablations in all environments. In PixelCartPole, we clearly see the benefit of iterative strategies on reward: LICORICE, LICORICE-DE and LICORICE-AC consistently increase in reward and converge at high levels, but LICORICE-IT converges at less than 50% of the optimal reward. We also see that LICORICE-IT also achieves higher concept error, indicating that it struggles to learn the larger distribution of concepts induced by a non-optimal policy. In DoorKey, all algorithms steadily increase in reward. However, we can see a dip at around 10^6 environment steps where we begin the second iteration for LICORICE, LICORICE-AC, and LICORICE-DE. Although LICORICE-IT achieves similar reward to LICORICE, LICORICE achieves lower concept error, which is beneficial for the goal of interpretability. Finally, in DynamicObstacles, LICORICE-AC lags behind the most in terms of both reward and concept error. This result indicates that the active learning component is most important for this environment.

We now inspect the performance at the last training iteration in Figure 10. Overall, we find that LICORICE performs better than or equal to the ablations on all environments. It enjoys the best reward performance on PixelCartPole, while performing similarly to LICORICE-IT on DoorKey and to LICORICE-IT and LICORICE-DE on DynamicObstacles. It exhibits the lowest concept error on PixelCartPole and DoorKey, and achieves similarly low error to LICORICE-DE and LICORICE-IT on DynamicObstacles.

In PixelCartPole, LICORICE achieves the highest reward, indicating the benefits of multiple iterations, active learning, and decorrelation, which are absent in LICORICE-IT, LICORICE-AC, and LICORICE-DE, respectively. LICORICE also enjoys the lowest concept error, showcasing its ability to learn and apply concepts accurately. LICORICE-AC and LICORICE-DE perform well in minimizing concept error, while LICORICE-IT shows a relatively higher error rate, underscoring the importance of multiple iterations in some environments.

B.5 Hyperparameter Sensitivity

We now seek to understand how sensitive LICORICE is to the choice of hyperparameters. Figure 11 shows the results of this experiment for DynamicObstacles. Interestingly, we find that the main difference between the settings is the variation. Overall, LICORICE is relatively robust to the choice of hyperparameter settings in this environment, with the final average reward ranging between 96%

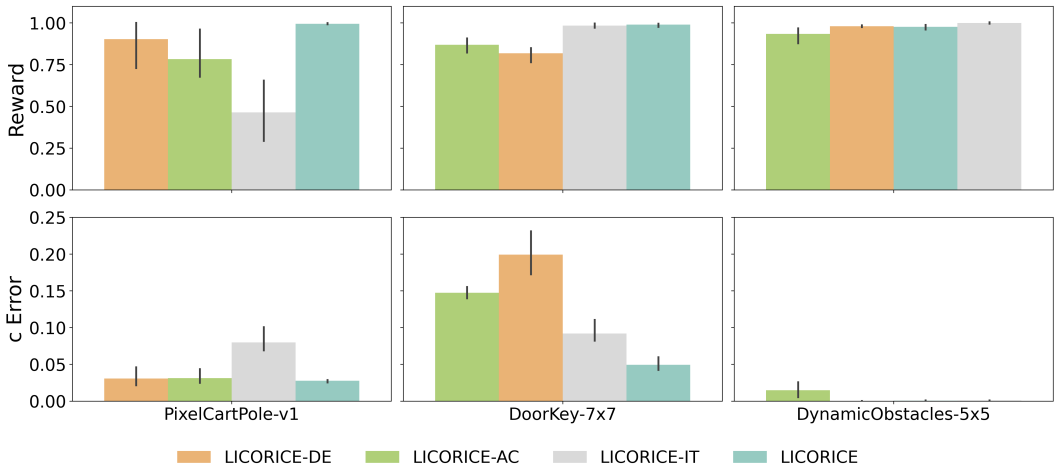

Figure 10: Performance of ablations at the final training iteration in the first three environments, with black bars representing the 95% confidence interval, calculated using 1000 bootstrap samples.

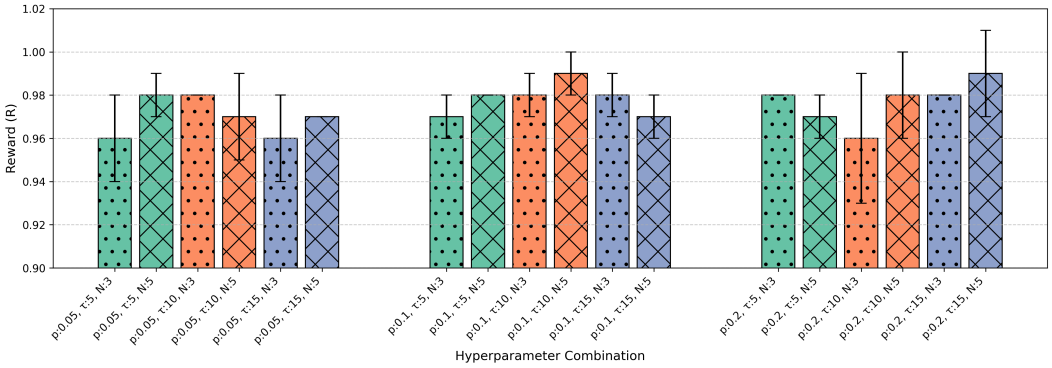

Figure 11: Hyperparameter sensitivity results for DynamicObstacles. Bars represent standard deviation. Here, bars are grouped by the same p values (acceptance probability). Bars of the same color share the same τ value (budget multiplier), and bars with the same pattern share the same value for N (committee size). Generally, $p = 0.1$ tend to perform better overall with greater stability. Having a larger committee ($N = 5$ vs. $N = 3$) also tends to help with performance.

and 99.5% of the optimal. The concept error for all settings was 0.00 ± 0.00, so we do not plot it here.

B.6 Choice of Acquisition Strategy

We now investigate the impact of the choice of acquisition strategy on both reward and concept accuracy. We implemented an entropy-based disagreement measurement for the committee and studying its effects on a MiniGrid environment. Specifically, we implement the following vote entropy acquisition function: $U(s_i) = -\frac{1}{K}\sum_{k=1}^{K}\sum_{j=1}^{P_k}\frac{V_{ijk}}{N}\log\frac{V_{ijk}}{N}$, where s_i is the input state, K is the number of concepts we learn, P_k is the number of classes for the k-th concept, V_{ijk} is the number of votes for the j-th class of the k-th concept for state s_i, and N is the number of committee members.

Table 9 shows these results on DynamicObstacles. There is essentially no difference between the two strategies in terms of both reward and concept error.

	100		200		300	
	R	c Error	R	c Error	R	c Error
Disagreement	0.96	0.05	0.97	0.01	1.00	0.00
Entropy	0.95	0.07	0.98	0.02	0.99	0.00

Table 9: Comparison of disagreement- and entropy-based active learning strategies for DynamicObstaclesunder different labeling budgets. There is essentially no difference between the two strategies.

C Additional Discussion

C.1 Limitations and Future Work

While our approach has demonstrated promising results, there are several limitations to be addressed in future work. One significant challenge is the difficulty VLMs face with certain types of concepts, especially continuous variables. This limitation can impact the overall performance of concept-based models, especially in domains where continuous data is prevalent. Furthermore, although VLMs can be successfully used for automatic labeling of some concepts, there are still hallucination issues (Achiam et al., 2023) and other failure cases, such as providing inaccurate counts. We believe that future work that seeks to improve general VLM capabilities and mitigate hallucinations would also help overcome this limitation. Addressing this issue could involve developing specialized techniques or using existing tools and libraries to better complement VLM capabilities.

Another area for future improvement is the refinement of our active learning and sampling schemes. Our current method employs an disagreement-based acquisition function to select the most informative data points for labeling. While this approach is effective, there is potential for exploring more sophisticated active learning strategies, such as incorporating advanced exploration-exploitation trade-offs or leveraging recent advancements in active learning algorithms (Tharwat & Schenck, 2023).

Furthermore, scaling challenges may arise from environmental complexity, such as when only a subset of given concepts are relevant or when learning certain concepts is prerequisite to gathering training data for others. Future research could explore using attention mechanisms (Vaswani et al., 2017) or sparse coding (Olshausen & Field, 1996) to identify relevant concepts. Another exciting direction is the work in human-AI complementarity and learning-to-defer algorithms (Mozannar et al., 2023) to train an additional classifier for deferring labeling to a person when the chance of an error is high. In our work, we assume a known set of concepts for the environment; future work could investigate the use of human-VLM teams to determine a reasonable concept set for the environment.

Finally, designing a concept-based representation for RL remains an open challenge. Our work provides a few illustrative examples, but the exact design of these representations can significantly impact performance, often for reasons that are not entirely clear — especially when using VLMs as annotators. Prior work (Das et al., 2023) proposed some desiderata for concepts in RL, but future work could refine these principles, especially in the face of VLM annotators. Future work could also include systematically investigating the factors that influence the effectiveness of different concept-based representations in RL. This could involve extensive empirical studies, theoretical analyses, and the development of new design principles that guide the creation of effective concept representations. Understanding these factors better will help in creating more reliable and interpretable RL models, ultimately advancing the field and broadening the applicability of concept-based approaches in various RL tasks.

C.2 Broader Impacts

Interpretability in RL Incorporating concept learning with RL presents both positive and negative societal impacts. On the positive side, promoting interpretability and transparency in decision-making fosters trust and accountability. However, in cases where it yields incorrect results, stakeholders might be misled into trusting flawed decisions due to the perceived transparency of the model (Kaur et al., 2020). Unintended misuse could also occur if stakeholders lack the technical expertise to accurately interpret the models, leading to erroneous conclusions and potentially harm-

ful outcomes. To mitigate these risks, an avenue for future work is developing clear guidelines for interpreting these models and tools to scaffold non-experts' understanding of the model outputs.

Using VLMs for Concept Labeling On one hand, VLMs have the potential to significantly improve the efficiency and scalability of labeling processes, which can accelerate advancements in various fields. By automating the labeling of large datasets, VLMs can help reduce the time and cost associated with manual labeling. However, there are important ethical and social considerations to address. One major concern is the potential for bias in the concept labels generated by VLMs. If these models are trained on biased or unrepresentative data, they may perpetuate or even amplify existing biases, leading to unfair or discriminatory outcomes. This is particularly problematic in sensitive applications like hiring, lending, or law enforcement, where biased decisions can have significant negative impacts on individuals and communities. Furthermore, there are privacy concerns related to the data used to train VLMs. Large-scale data collection often involves personal information, and improper handling of this data can lead to privacy violations. To mitigate these risks, future work could include developing robust data governance frameworks to protect individuals' privacy and comply with relevant regulations.