EMBODIED ACTIVE LEARNING OF RELATIONAL STATE ABSTRACTIONS FOR BILEVEL PLANNING

Amber Li
MIT CSAIL
amli@alum.mit.edu

Tom Silver
MIT CSAIL
tslvr@mit.edu

ABSTRACT

State abstraction is an effective technique for planning in robotics environments with continuous states and actions, long task horizons, and sparse feedback. In object-oriented environments, predicates are a particularly useful form of state abstraction because of their compatibility with symbolic planners and their capacity for relational generalization. However, to plan with predicates, the agent must be able to interpret them in continuous environment states (i.e., ground the symbols). Manually programming predicate interpretations can be difficult, so we would instead like to learn them from data. We propose an embodied active learning paradigm where the agent learns predicate interpretations through online interaction with an expert. For example, after taking actions in a block stacking environment, the agent may ask the expert: "Is On(block1, block2) true?" From this experience, the agent *learns to plan*: it learns neural predicate interpretations, symbolic planning operators, and neural samplers that can be used for bilevel planning. During exploration, the agent *plans to learn*: it uses its current models to select actions towards generating informative expert queries. We learn predicate interpretations as ensembles of neural networks and use their entropy to measure the informativeness of potential queries. We evaluate this approach in three robotic environments and find that it consistently outperforms six baselines while exhibiting sample efficiency in two key metrics: number of environment interactions, and number of queries to the expert.

1 INTRODUCTION

Our research objective is to develop a robotic agent that can achieve a wide variety of high-level goals, like preparing a meal or cleaning up a kitchen, in environments with continuous state and action spaces, long task horizons, and complex constraints. In this work, we study an *embodied active learning* paradigm, where the robot learns by interacting with its environment, querying expert knowledge, and using the expert's feedback to guide its subsequent exploration and queries (Daniel et al., 2014). Since real-world exploration and data collection is expensive, we want the robot to 1) minimize the number of actions taken in the environment and 2) ask the expert as few questions as possible. In other words, the agent must select actions and query strategically.

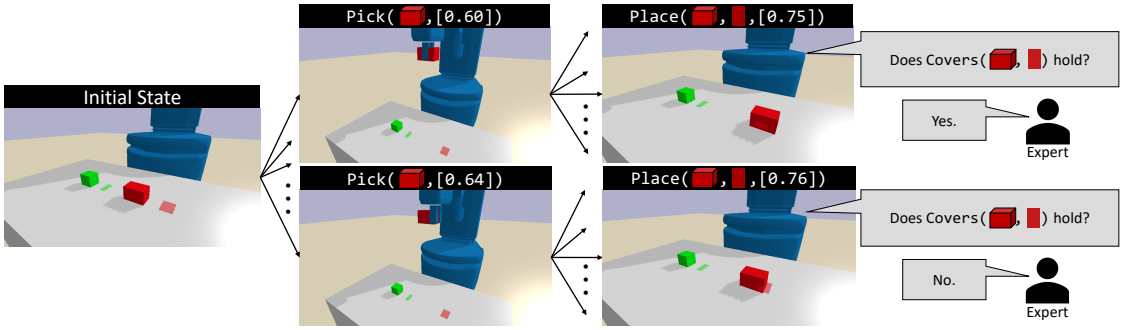

Figure 1: **Active predicate learning in the PickPlace1D environment**. The `Covers` predicate is difficult to interpret: given limited data, the agent may not know how to classify a block partially overlapping a region. To improve its interpretations, the agent must take actions to set up classification problems and then query the expert for labels. The figure shows two possible trajectories among infinitely many. There are also queries at intermediate states (not shown).

Towards achieving a wide distribution of goals in robotic environments, we consider an agent that is *learning to plan*. In particular, we build on *task and motion planning (TAMP)* (Garrett et al., 2020), which uses state and action abstractions to plan efficiently in continuous environments. Previous work has shown how to learn action abstractions (operators and samplers) when *given* state abstractions (predicates) for TAMP (Silver et al., 2021; Chitnis et al., 2022). However, hand-specifying the state abstractions can be tedious and impractical, even for an expert programmer. In this work, we consider the problem of *learning* these state abstractions via embodied active learning. State abstractions in TAMP take the form of *predicates*. A predicate is a named relation over objects, and the semantic interpretation of a predicate is defined by a binary classifier. For example, in the PickPlace1D environment (Silver et al., 2021; Chitnis et al., 2022) (Figure 1), a predicate called `Covers` takes two object arguments, a `block` and a `target`, and the associated classifier returns true if the block completely covers the target. Applying a set of predicate classifiers to a continuous state induces a discrete abstract state, e.g., $\{$`Covers(b1, t1)`, `HandEmpty(rob)`, ...$\}$. Given a predicate-based goal, TAMP searches in the abstract state space to constrain search in the continuous state space.

We propose *active predicate learning for TAMP*. A robot is situated in a deterministic environment with an expert. To begin, the expert gives a small number of demonstrations (to illustrate the task distribution of interest) and a very small number of classification examples (one positive and one negative) for each predicate. At this point, the robot knows the predicates but not their interpretations; in other words, it needs to solve a symbol grounding problem (Harnad, 1990)[1]. The robot starts to *explore* its environment: at each step, the robot selects an *action* to execute and a *query* to give the expert. The query is a set of zero or more ground atoms (predicates with object arguments) that the robot wants to "check" in the current state. For example, querying $\{$`Covers(b1, t1)`$\}$ would ask if `b1` currently covers `t1` according to the expert's interpretation. The expert answers "yes" or "no" according to a noise-free but unknown ground-truth interpretation. To deal with possible dead-ends, the expert also periodically resets the environment to an initial state drawn from a distribution. This setting is reminiscent of how a young child might use very sparse linguistic labels in early concept learning (Bowerman et al., 2001; Casasola & Bhagwat, 2007). To measure the extent to which the robot uses its experience to improve its planning ability, we evaluate the robot on a set of held-out planning tasks.

In this setting, the agent is faced with two interrelated subproblems: how to query, and how to select actions. For example, towards learning the meaning of `Covers`, querying about a block that partially overlaps a target may be more informative than querying about a block that is far from a target. Furthermore, the agent may need to carefully select a grasp and place position to reach an "interesting" state where there is partial overlap to ask about (Figure 1). This need for action selection is what distinguishes embodied active learning from typical active learning (Settles, 2011), and the availability of an expert to query distinguishes the setting from exploration in (model-based) reinforcement learning (Kaelbling et al., 1996). Nonetheless, we can draw on both of these lines of work to make progress here.

We propose an *action selection strategy* and a *query policy* for active predicate learning. Both are rooted in the active learning principle that the robot should reduce its uncertainty about its classifiers. The query policy selects ground atoms whose classification entropy is above a certain threshold. Action selection uses the robot's learned predicates, operators, and samplers to *plan* to reach states where there is high entropy. In experiments, we compare against alternative action selection and query policies and find that our main approach effectively balances action cost (number of environment transitions) and query cost (number of ground atoms asked). In summary, we (1) propose the problem setting of active predicate learning for TAMP; (2) propose an entropy-based, model-based approach; and (3) evaluate the approach in simulated robotic environments.

2 Problem Setting

Environments. We consider a robot exploring an environment with deterministic transitions and fully-observed states. A state $x \in \mathcal{X}$ is defined by a set of objects $\mathcal{O}$ and a real-valued feature vector for each object. The dimensionality of an object's feature vector is determined by the object's *type* $\lambda \in \Lambda$. For example, an object of type `block` may have a feature vector of dimension 4 describing its current pose (x, y, and z coordinates) and color. An action $u \in \mathcal{U}$ is a controller with discrete and continuous parameters. For example, `Pick(b1,[0.3,0.2,0.4])` is an action for picking block `b1` with continuous grasp pose `[0.3,0.2,0.4]`. A deterministic simulator $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ predicts a next state given a current state and action. The simulator is known to the robot[2], who can use it to plan. An environment can be viewed as a form of object-oriented (Diuk et al., 2008) or relational (Guestrin et al., 2003) MDP, but note the continuous states and hybrid discrete-continuous actions.

Predicates. A *predicate* ψ consists of a name (e.g., `Covers`) and a tuple of typed placeholders for objects $(\lambda_1, \ldots, \lambda_m)$ (e.g., `(?block, ?target)`). The *interpretation* of a predicate is a classifier $c_\psi : \mathcal{X} \times \mathcal{O}^m \to$

[1] Another aspect of symbol grounding, which we do not address here, is generating referents for objects.

[2] Previous work by Chitnis et al. (2022) has shown that this simulator can also be learned.

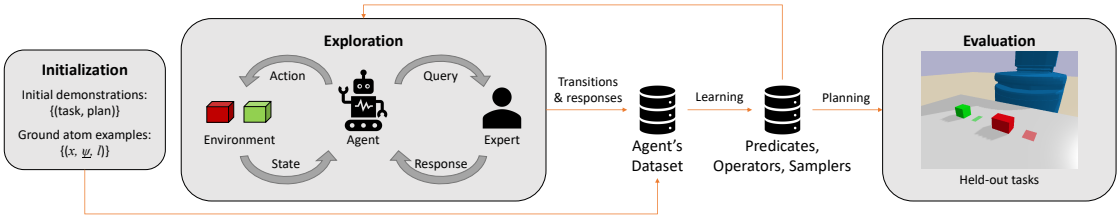

Figure 2: **Problem setting and approach overview.** The agent's dataset is initialized with a small number of demonstrations and ground atom examples, which it uses to learn initial predicates, operators, and samplers. Those models are then used during exploration, where the agent takes actions in the environment and queries the expert. From these interactions, the dataset grows and the models improve. We periodically evaluate the agent on held-out planning tasks.

$\{\text{true}, \text{false}\}$. These classifiers are unknown to the agent and must be learned. We distinguish between ground atoms, where a predicate is applied to specific objects $(o_1, \ldots, o_m)$, and lifted atoms, where the predicate is applied to typed variables. For example, `Covers(b1, t1)` is a ground atom and `Covers(?block, ?target)` is a lifted atom. The interpretation of a ground atom $\underline{\psi}$ with objects $\bar{o} = (o_1, \ldots, o_m)$ is given by $c_{\underline{\psi}}(x) := c_{\psi}(x, \bar{o})$.

Initialization. Before exploration, the robot is presented with a small set of demonstrations. Each demonstration consists of a *task* and a *plan*. The task consists of an initial state $x_0 \in \mathcal{X}$ and a goal g. The goal is a set of ground atoms and is said to *hold* in a state x if $c^*_{\underline{\psi}}(x) = \text{true}$ for all ground atoms $\underline{\psi} \in g$, where $c^*_{\underline{\psi}}$ is the (unknown) expert interpretation of $\underline{\psi}$. A plan is a sequence of actions $\pi^* = (u_1, \ldots, u_n)$. The plan need not be optimal, but it is assumed to solve the task, i.e., simulating π^* forward from x_0 will terminate at a state where g holds. The expert additionally presents a very small[3] set of examples $\mathcal{D} = \{(x, \underline{\psi}, \ell)\}$, where $\ell \in \{\text{true}, \text{false}\}$ is the output of $\underline{\psi}(x)$ under the expert's interpretation. This dataset communicates the full set of predicates Ψ that the robot can use to query the expert. In other words, the expert assumes that the robot will extract the predicates in $\mathcal{D}$ – `Covers`, `Holding`, and so on – and use them to form queries.

Exploration and evaluation. After initialization, the robot begins to explore the environment. At the start of each *episode* of exploration, the environment is reset to an initial state $x_0 \in \mathcal{X}$ sampled from an initial state distribution. Then, for up to h steps, the robot repeatedly queries the expert about the current state x and executes an action to advance the state x'. A *query* $\mathcal{Q}$ is a set of ground atoms, and a *response* is a set of $(\underline{\psi}, \ell)$ where $\underline{\psi} \in \mathcal{Q}$ and $\ell = c^*_{\underline{\psi}}(x)$. An example query in PickPlace1D is $\{$`Covers(b1, t1)`, `Covers(b1, t2)`$\}$, and a possible response is $\{$(`Covers(b1, t1)`, True), (`Covers(b1, t2)`, False)$\}$. Each response is added to the robot's dataset with the current state, i.e., $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x, \underline{\psi}, \ell)\}$. To measure progress, we periodically *evaluate* the robot on a set of held-out tasks. Each task $\langle x_0, g \rangle$ is considered solved if the robot reaches g from x_0 within h steps and within a planning timeout. The robot's objective is to take actions, make queries, and use the responses to maximize the number of tasks solved after a minimal number of exploration actions and queries.

3 Learning Abstractions for Bilevel Planning

Our work builds on recent advances in learning abstractions for bilevel planning (Silver et al., 2021; Chitnis et al., 2022), a specific instantiation of TAMP. We review the key ideas here and refer readers to the references for details.

3.1 Bilevel Planning with Predicates, Operators, and Samplers

The first key idea in bilevel planning is that *predicates induce abstract states*. In particular, given a state x, a set of predicates Ψ, and their interpretations, we can create a corresponding abstract state

$$\texttt{abstract}(x, \Psi) := \{\underline{\psi} : c_{\underline{\psi}}(x) = \text{true}\}.$$

We use s to denote an abstract state. An abstract state will generally lose information about the original state. However, if the predicates are defined judiciously, they can provide guidance for planning.

Operator-PlaceToCover:
Arguments: `[?b - block, ?t - target, ?r - robot]`
Preconditions: `{Holding(?b)}`
Add Effects: `{Covers(?b, ?t), HandEmpty(?r)}`
Delete Effects: `{Holding(?b)}`
Controller: `Place(?b, ?t, ?r, 0)`

[3] In experiments, we use one positive and one negative example per predicate.

The second key idea is that *abstract actions define transitions between abstract states.* Abstract actions are defined in terms of *operators* and *samplers*. An operator has arguments, preconditions, effects, and a controller. We eschew formal definitions (Silver et al., 2021; Chitnis et al., 2022) in favor of simplified exposition and refer to the example on the right. The arguments are variables, i.e., typed placeholders for objects. The preconditions are lifted atoms that define what must be true in an abstract state for this operator to be applied. The effects determine how the abstract state would change if this operator were applied; add effects are added, and delete effects are removed. Finally, the controller connects the abstract action to the environment action space. The discrete parameters of the controller (e.g, which target to place on) are determined in the operator, but the continuous parameters (e.g., what position offset to use) are undetermined. To propose different values for the continuous parameters, a *sampler* is associated with the operator. A *ground operator* is an operator whose arguments have been substituted for objects. The assignment of arguments to objects is also given to the sampler, along with the current state, so that the sampler can propose targeted values for the controller. We use $a \in \mathcal{A}$ to denote a ground operator and $F(s, a) = s'$ to denote the (partial) *abstract transition function* induced by the operators.

Given predicates, operators, samplers, and a task $\langle x_0, g \rangle$, bilevel planning generates candidate *abstract plans* and then attempts to *refine* those plans into environment actions (Silver et al., 2021; Chitnis et al., 2022). An abstract plan comprises a *subgoal sequence* and a *plan sketch*. The subgoal sequence consists of abstract states $(s_0, \ldots, s_n)$ where $s_0 = \texttt{abstract}(x_0, \Psi)$ and $g \subseteq s_n$. For example, if $s_1 = \{\texttt{Holding(rob, b1)}\}$, then the robot will attempt to find a continuous action that leads to it holding `b1`. The plan sketch is a sequence of ground operators $(a_1, \ldots, a_n)$ and their associated samplers. Abstract plans are generated iteratively using an AI planner (Hoffmann, 2001; Helmert, 2006). For each abstract plan, the samplers in the plan sketch are repeatedly invoked until all subgoals are reached, or until a maximum number of tries is exceeded, at which point the next abstract plan is considered.

3.2 Learning Operators and Samplers Given Predicates

Our focus in this work is on learning predicate interpretations through embodied active learning. Given predicate interpretations, previous work has shown how to learn operators (Silver et al., 2021) and samplers (Chitnis et al., 2022). We use these techniques without modification and describe them very briefly here.

Given a dataset of transitions (x, u, x'), we can create a corresponding dataset of abstract state transitions (s, u, s') where $s = \texttt{abstract}(x, \Psi)$ and $s' = \texttt{abstract}(x', \Psi)$. To learn operators, the latter dataset is first partitioned so that two transitions are in the same partition set if their controllers and effects (changes in abstract state) are equivalent up to object substitution. For example, all transitions where a `Place` controller was used to successfully achieve `Covers` would be grouped together. For each partition set, preconditions are determined by finding all atoms in common at the start of each transition, again up to object substitution. This can be calculated efficiently via set intersection after objects are replaced with variable placeholders, which in turn become arguments. With arguments, preconditions, effects, and controllers determined for each partition set, the operators are complete.

The partition of abstract transitions is used once more for sampler learning. Each sampler is implemented as a neural network that takes in the object features of the operator arguments and returns the mean and diagonal covariance of a multivariate Gaussian distribution. The Gaussian is then used to propose values for the controller parameters. Training data for each sampler is extracted from the respective partition set and the neural networks are trained to minimize a Gaussian negative log-likelihood loss.

4 Active Predicate Learning

We want the robot to explore *efficiently* so that it can maximize performance on our real objective: *effectively* solving held-out planning tasks. As the robot explores, its dataset of environment transitions and query responses will grow. How should it make use of these data? Where and what should it explore next? We propose that the robot should *learn to plan* and then *plan to explore*.

4.1 Neural Predicate Learning

Recall that a dataset $\mathcal{D}$ of query responses $(x, \underline{\psi}, \ell)$ is given to the robot during initialization and then extended during exploration. We use these data to train *neural predicate classifiers* (interpretations). Each classifier c_ψ is parameterized as an ensemble of k fully-connected neural networks $h^{(1)}_{c_\psi}, \ldots, h^{(k)}_{c_\psi}$. Each member of the ensemble $h^{(i)}_{c_\psi}$ maps a state x and objects $\overline{o} = (o_1, \ldots, o_m)$ to a probability that the class is true. Since the full object set $\mathcal{O}$ can vary in size between tasks, we make the simplifying assumption that the only objects relevant to a predicate interpretation are those present

in the arguments[4]. We then parameterize each ensemble member as

$$h^{(i)}_{c_\psi}(x[o_1] \oplus \cdots \oplus x[o_m]),$$

where $x[o]$ denotes the feature vector of o in x and $\oplus$ denotes concatenation. The final output of the classifier $c_\psi(x)$ is true if the average predicted probability of the ensemble members exceeds 0.5.

Since the predicate arguments are typed and feature dimensions are fixed per type, the input to each ensemble member is a fixed-dimensional vector. Thus, if we can construct input-output training examples from $\mathcal{D}$, we can use standard neural network classifier training techniques. To construct these examples, we partition $\mathcal{D}$ into predicate-specific datasets, where the dataset for predicate ψ consists of $(x, \overline{o}, \ell)$ tuples where $(x, \underline{\psi}, \ell) \in \mathcal{D}$ and $\overline{o} = (o_1, \ldots, o_m)$ are the objects used to ground ψ in $\underline{\psi}$. For example, if $\psi =$ `Covers` and $\underline{\psi} =$ `Covers(b1, t1)`, then $\overline{o} =$ `(b1, t1)`. We further transform each $(x, \overline{o}, \ell)$ tuple into an input vector $x[o_1] \oplus \cdots \oplus x[o_m]$ and output class ℓ. With these data, we optimize the weights of $h^{(i)}_{c_\psi}$ to minimize binary cross-entropy loss via Adam (Kingma & Ba, 2014).

We use ensembles of neural networks because they provide a measure of uncertainty, which we will later leverage during exploration. (Other uncertainty quantification strategies are possible.) To this end, it is important that the networks converge to different hypotheses when there are many possible explanations of limited data. One way to promote diversity between networks is to initialize their weights differently at the start of training. In preliminary experiments, we found that increasing the variance of weight initialization led to greater ensemble disagreement but also convergence failures if the variance was too high. In our main experiments, we initialize network weights via a unit Gaussian distribution, detect convergence failures, and restart training if necessary; see Appendix A.1 for details.

With predicate interpretations learned, we can then apply the techniques from previous work to learn operators and samplers (Section 3.2). This whole training pipeline is executed for the first time during the initialization phase of active predicate learning (Section 2) and repeated after each episode of exploration. Given predicates, operators, and samplers, we have all the components needed for planning (Section 3.1). We can use this ability not only to solve held-out problems during evaluation, but also to guide exploration.

4.2 Model-based Exploration

Given predicates, operators, and samplers learned from the data collected so far, how should the robot collect more data to improve these models? To answer this question, we must define mechanisms for (1) query generation and (2) sequential action selection. In generating queries, the robot should reason about the value of different possible queries and trade off the need to collect more data with the cost of burdening the expert. In selecting actions, the robot should seek out regions of the state space where it can gather the most information to improve its models.

4.2.1 Query Generation

When the robot is in a state and deciding what queries to give the expert, it is solving an *active learning* problem. One of the main principles in active learning is that queries should be selected on the basis of the robot's *epistemic uncertainty* about potential responses. For example, if the robot is confident that `GripperOpen(rob)` is true and `Holding(b1, rob)` is false in the current state, then neither ground atom would be worth including in a query. If the robot is more unsure about `Covers(b1, t1)`, then that ground atom would be a better choice.

We use classifier *entropy* as a measure of epistemic uncertainty. Let $P(c_{\underline{\psi}}(x) = \ell) = \frac{1}{k}\sum_{i=1}^{k} P(h^i_{\underline{\psi}}(x) = \ell)$ denote the probability that the interpretation of ground atom $\underline{\psi}$ is ℓ in state x according the robot's current ensemble. The entropy for $\underline{\psi}$ in x is then

$$\texttt{entropy}(\underline{\psi}, x) := -\sum_{\ell=0,1} \left(P\left(c_{\underline{\psi}}(x) = \ell\right)\right) \log\left(P\left(c_{\underline{\psi}}(x) = \ell\right)\right).$$

We use entropy to define a *query policy*:

$$\pi_{\text{query}}(x) = \{\underline{\psi} : \texttt{entropy}(\underline{\psi}, x) > \alpha, \forall \underline{\psi} \in \underline{\Psi}\},$$

where α is a hyperparameter ($\alpha = 0.05$ in experiments) and $\underline{\Psi}$ is the set of all ground atoms. This query policy dictates that the robot will ask the expert about all ground atoms whose interpretations in the current state are sufficiently uncertain. The policy is *greedy* in the sense that it only uses the robot's current uncertainty, rather than predicting how its uncertainty would change given different responses (Settles, 2011). Nonetheless, as the robot collects more data and revises its predicate interpretations, its uncertainty will generally decrease, and the number of ground atoms included in queries will generally decline.

[4]In general, this assumption is limiting. Relational neural networks (e.g., GNNs) may be used to avoid this assumption.

4.2.2 Sequential Action Selection

After the robot generates a query and receives a response, it must select an action to take. In practice, the robot will make an entire plan and then generate a query at each of the states it encounters. The main consideration in action selection is that the robot should visit states that allow for informative queries. For example, in the PickPlace1D environment (Figure 1), the robot may be very uncertain about the interpretation of `Covers` in the case where a block is overlapping, but not completely covering, a target region. Since blocks are always disjoint from targets in initial states, the robot would need to carefully select a `Pick` and a `Place` action before it can ask the expert about this case.

Algorithm 1 Lookahead Action Selection

1: **inputs:** state x_0, predicates Ψ, ground operators $\mathcal{A}$,
2: learned samplers Ω, simulator f
3: **hyperparameters:** `maxTrajs`, `maxHorizon`
4: **repeat** `maxTrajs` times
5: $x \leftarrow x_0$
6: `score` $\leftarrow 0$
7: **repeat** `maxHorizon` times
8: $s \leftarrow$ `abstract`(x, Ψ)
9: $a \leftarrow$ `sampleApplicableOp`$(s, \mathcal{A})$
10: $u \leftarrow$ `sampleAction`(a, x, Ω)
11: $x \leftarrow f(x, u)$
12: `stateScore` $\leftarrow \sum_{\underline{\psi}}$ `entropy`$(\underline{\psi}, x)$
13: `score` $\leftarrow$ `score` $+$ `stateScore`
14: **end**
15: **end**
16: **return** action trajectory that maximized `score`

Since the robot is learning models for bilevel planning, a natural question is whether we can leverage these models for action selection during exploration. Previous work on *Goal-Literal Babbling (GLIB)* has shown that planning to achieve randomly sampled goals can be an effective strategy for online operator learning in the case where predicates are known (Chitnis et al., 2021). However, since goals are discrete atoms, GLIB is unable to pursue specific low-level states. For example, even if GLIB sampled a goal with `Covers(b1, t1)`, the bilevel planner described in Section 3.1 would have no mechanism to seek out an information-rich state where `b1` is partially overlapping `t1`. Thus, this new problem setting where we are learning not only operators but also predicates (and samplers) through online interaction calls for a different action selection strategy.

We propose a *lookahead* action selection strategy that uses the robot's current models for planning while taking into account the information value of candidate low-level states. The strategy is summarized in Algorithm 1. Given an initial state and the robot's current predicates, operators, and samplers, the robot samples and simulates `maxTrajs` possible trajectories. Each trajectory is sampled by repeatedly abstracting the state using the learned predicate interpretations (Line 8), sampling a learned operator whose preconditions hold (Line 9), sampling an action using the learned samplers (Line 10), and advancing the state (Line 11). Each state encountered is scored according to the total entropy over all ground atoms (Line 12), and each trajectory is scored by accumulating these scores over all encountered states. Finally, the actions from the trajectory with the highest score are selected for execution in the environment (Line 16). In practice, in the case where no applicable operators can be found, we terminate the trajectory early. Furthermore, in the case where no nontrivial trajectory can be found, we fall back to sampling a random action (Chitnis et al., 2021).

This lookahead action selection strategy is closely tied to query generation: the robot will seek out states with high entropy, and then query the expert to reduce its uncertainty in those states. We hypothesize that this tight relationship is essential for efficient active predicate learning. Furthermore, we hypothesize that exploring to reduce predicate entropy will sufficiently drive operator and sampler learning[5]. To test these hypotheses, we turn to experiments.

5 Experiments

We now present experimental results evaluating the extent to which our main approach effectively and efficiently learns predicates useful for bilevel planning. We evaluate seven approaches (main and six baselines) in three environments.

5.1 Experimental Setup

Approaches. Here we briefly describe the approaches, with additional details in Appendix A.1. In addition to our main approach, we consider three action selection baselines and three query generation baselines.

- **Main.** Our main approach, which uses lookahead action selection and the entropy-based query policy.
- **GLIB.** Same as Main except GLIB (Chitnis et al., 2021) is used for action selection instead of lookahead.

[5]However, in cases where the predicate interpretations are easy to learn, or already known, additional exploration techniques (e.g., GLIB (Chitnis et al., 2021)) may be useful to drive operator and sampler learning.

- **Random Actions.** Same as Main except actions are selected uniformly at random.
- **No Actions.** Same as Main except no actions are taken during exploration (only initial states are queried).
- **Ask All.** Same as Main except all possible queries are generated at every step of exploration.
- **Ask None.** Same as Main except no queries are generated.
- **Ask Randomly.** Same as Main except queries are selected uniformly at random from the set of all possible ground atoms. The number of ground atoms in the query is approximately equal to the number generated by the main entropy-based query policy.

Environments. We now briefly describe the environments, with additional details in Appendix A.1.

- **PickPlace1D.** As described in Section 1, this environment features a robot that must pick blocks and place them to completely cover target regions along a table surface. All pick and place poses are in a 1D line. Evaluation tasks require 1–4 actions to solve. The predicates are `Covers`, `Holding`, and `HandEmpty`. This environment was proposed by Silver et al. (2021), who used manually designed predicates.
- **Two Rooms.** This is a novel environment that is very loosely inspired by the continuous playroom of Konidaris & Barto (2009). Two rooms are connected by a hallway. One room has a table with 3 blocks; the other room has a continuous dial for turning a light on or off. Evaluation tasks require 4–8 actions to solve. The predicates are `On`, `OnTable`, `GripperOpen`, `Holding`, `Clear`, `NextToTable`, `NextToDial`, `LightOn`, `LightOff`.
- **Blocks.** This is a robotic version of the classic blocks world environment. During exploration, 3 or 4 blocks are available. Evaluation tasks have 5 or 6 blocks and require 2–20 actions to solve. The predicates are `On`, `OnTable`, `GripperOpen`, `Holding`, `Clear`. This environment was also used by Silver et al. (2021) with manually designed predicates.

Experimental details. All approaches are run across all environments for 1000 transitions and evaluated after every episode on 50 held-out evaluation tasks. Each trial is repeated over 10 random seeds. Our key metrics are (1) number of evaluation tasks solved within a planning timeout (10 seconds) and (2) cumulative query cost (total number of ground atoms asked). Exploration episode lengths are 3, 8, and 20 steps for PickPlace1D, Two Rooms, and Blocks respectively. Demonstrations in the initial dataset are generated with environment-specific scripts (50 per environment). Query responses are generated automatically via scripted predicate interpretations. The initial dataset includes 1 positive and 1 negative example, selected randomly, of each predicate in each environment. All experiments were conducted on a quad-core Intel Xeon Platinum 8260 processor. See Appendix A.1 for additional experimental details.

5.2 Results & Discussion

Our main results are shown in Figure 3. Comparing the Main approach to **the action selection baselines**, we first see that the number of evaluation tasks solved quickly exceeds that of the No Action baseline. This confirms that there is value in exploring beyond initial states and that this embodied active learning setting is meaningfully different from standard active learning. The Main approach is also far more sample-efficient than Random Actions, supporting the hypothesis that directed exploration is important for active predicate learning. Finally, we see that Main outperforms GLIB in PickPlace1D and Blocks and performs similarly in Two Rooms. These results suggest that exploring via planning to reach specific low-level states, like the partial overlaps in PickPlace1D discussed in Figure 1, can lead to efficiency gains versus exploring only in the abstract space. Furthermore, the main lookahead action selection strategy is benefiting from its direct connection to query generation: it considers predicate classifier entropy and does so for every state in the trajectory. Nonetheless, we believe that GLIB is better able to target goals that are far from the current state than the lookahead action strategy, which relies on random forward sampling. This may explain GLIB's strong performance in Two Rooms, where the agent should take multiple actions to move from one room to another during exploration. Combining the strengths of GLIB and lookahead is an exciting direction for future work.

We next compare the Main approach to **the query generation baselines**. As expected, the Ask None approach performs very poorly because the initial dataset does not contain sufficient class labels to learn good predicate interpretations. The Ask All approach performs similarly to Main in terms of evaluation tasks solved but much worse in terms of cumulative query cost. Ask All continues to accumulate enormous query costs throughout exploration; Main generates a modest number of queries in the beginning, when the agent's uncertainty is high, before plateauing to near "silence" when its uncertainty is low. This confirms that the Main approach is querying enough to learn effectively while avoiding unnecessary queries (e.g., burdening a human expert). Interestingly, in PickPlace1D, Main seems to

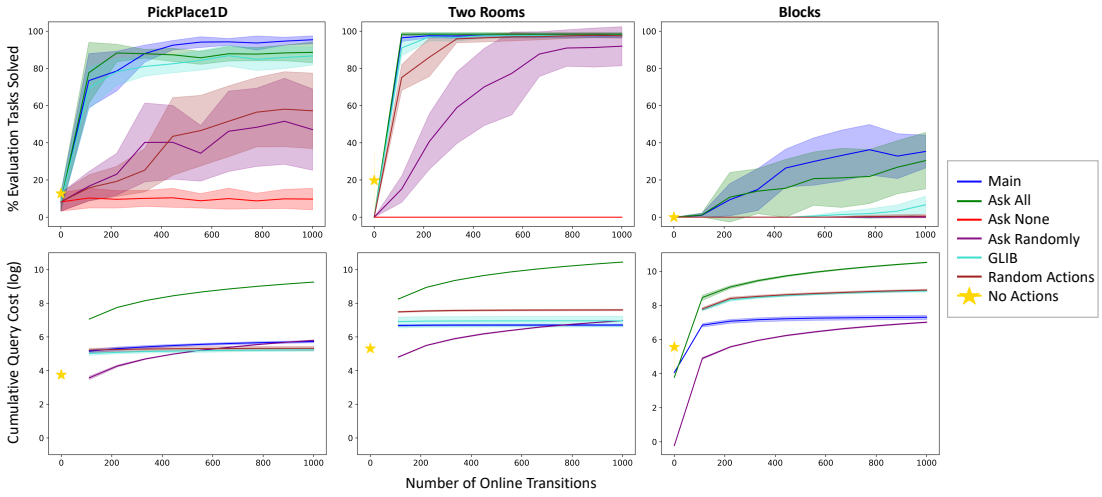

Figure 3: **Main results.** Our main approach solves the same number of evaluation tasks as Ask All (top) using far fewer expert queries (bottom). Query cost is the total number of ground atoms included in queries over time. All results are averaged over 10 random seeds. Lines are means and shaded regions are 95% t-confidence intervals[7]. Note that No Actions is a single point at 0.

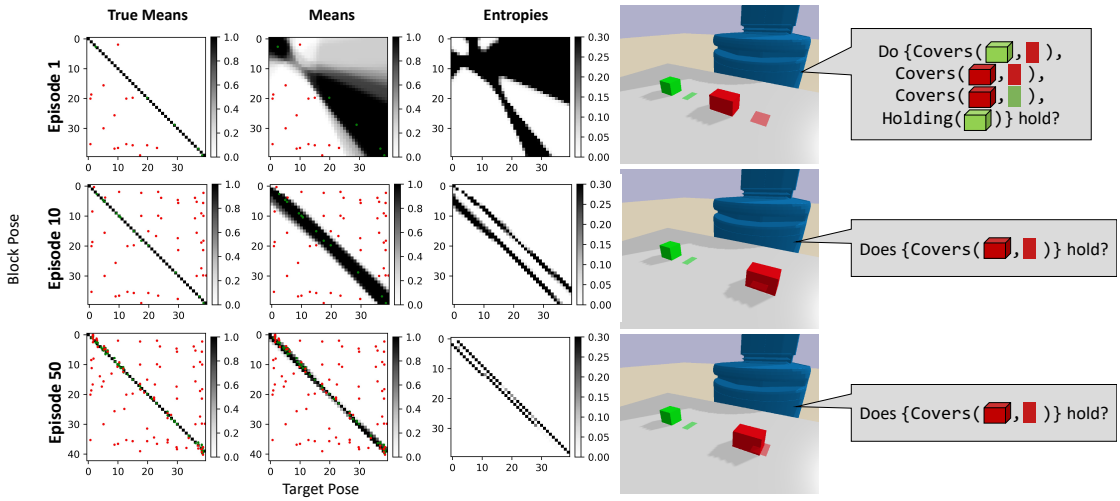

Figure 4: **Active predicate learning example for a single seed in PickPlace1D.** (Left) The left column (True Means) shows the ground-truth interpretation of the `Covers` predicate, which holds true (black) when the difference between the target pose and block pose is less than a small threshold. The middle column (Means) shows the agent's predictions averaged over the ensemble, and the right column (Entropies) shows the entropies. Red dots are negative examples and green dots are positive examples. As exploration progresses (top to bottom), the agent makes queries in high-entropy regions and learns better interpretations. (Right) The queries become smaller and more targeted over time.

slightly outperform Ask All in terms of evaluation tasks solved, though the confidence intervals overlap slightly. Inspecting the data collected by Ask All, we find a high density of points in regions of input space that are far from the boundary between positive and negative classification; intuitively, we believe these points "distract" training from the edge cases that Main is able to "focus" on, given its more targeted dataset. Finally, comparison to the Ask Randomly approach shows that the Main approach is selecting its queries judiciously.

Figure 4 illustrates the Main approach in the PickPlace1D environment. On the left, we see that entropy for the `Covers` classifier is initially high in large regions of the input space. After 10 episodes of exploration, the entropy

[7]The sample size is $n = 10$ (the number of random seeds).

PickPlace1D			Two Rooms									Blocks				
Hand	Cover	Hold	NDial	Open	On	Hold	NTab	LOff	OnTab	Clear	LOn	Hold	Open	OnTab	Clear	On
5.5%	84.4%	10.1%	5.0%	5.3%	44.8%	13.9%	4.1%	1.7%	11.5%	12.0%	1.7%	3.8%	0.1%	12.5%	3.4%	80.2%

Table 1: **Query percentages per predicate for Main approach.** Table entries are means over 10 seeds. The predicates from left to right are `HandEmpty`, `Covers`, `Holding`, `NextToDial`, `GripperOpen`, `On`, `Holding`, `NextToTable`, `LightOff`, `OnTable`, `Clear`, `LightOn`, `Holding`, `GripperOpen`, `OnTable`, `Clear`, `On`. Predicates with more difficult interpretations are generally included in more queries. For example, in Two Rooms, the `On` predicate is queried the most and the `LightOn` predicate is queried the least. Interpreting `On` requires relating the 3D poses of two blocks, while interpreting `LightOn` only requires a threshold on a single feature of the light.

is much more concentrated around the diagonal of the input space, where the block is partially overlapping the target. By episode 50, the agent has repeatedly explored states with partial overlaps and refined its classifier further. On the right, we see that query generation becomes more focused as the classifiers improve. By episode 50, the agent queries almost exclusively about `Covers` in states with partial overlap, and evaluation performance is nearly perfect.

In Table 1, we analyze the number of queries asked per predicate and find that more difficult predicates are asked about more often. For example, the `Covers` predicate dominates the query cost in PickPlace1D, and the `On` predicate, which requires learning a function that relates the 3D poses of two blocks, is queried about most in Two Rooms and Blocks.

Appendix A.4 reports additional experimental findings. When we ablate away the MLP ensemble used for modeling predicate classifiers, performance degrades, confirming the importance of modeling epistemic uncertainty. When we inspect failures to solve evaluation tasks, we see two kinds: failure to find a plan within the timeout, and failure to achieve the goal even when a plan is found due to incorrect goal predicate interpretations. Finally, we analyze the `Covers` predicate from PickPlace1D on four illustrative classification examples.

6 Related Work

We now discuss connections to prior work.

Learning state abstractions for TAMP. This work contributes to the literature on learning state abstractions for TAMP and decision making more broadly (Li et al., 2006; Jetchev et al., 2013; Abel et al., 2016; Konidaris et al., 2018; Xu et al., 2020; Akakzia et al., 2021; Wang et al., 2021; Ahmetoglu et al., 2022; Migimatsu & Bohg, 2022). Particularly relevant is recent work by Silver et al. (2023) who consider learning predicates within their bilevel planning framework (Silver et al., 2021; Chitnis et al., 2022; Silver et al., 2022a). Our work is different and complementary in several ways: we learn the interpretations of *known predicates* from *interaction* with an expert in an *online setting*; they learn *latent predicates* from *demonstrations* in an *offline setting*. Furthermore, they make two key restrictions: predicate classifiers are implemented as simple programs (Pasula et al., 2007), and a small set of "goal predicates" are given. Since we have supervision for predicate learning, we are able to instead learn neural network predicate classifiers without given goal predicates. A straightforward combination of the two would use our approach to learn a small set of predicates from interaction, and their approach to invent additional predicates to aid in planning.

Exploration in relational domains. Our action selection strategy takes inspiration from previous work on exploration in relational domains (Walsh, 2010; Rodrigues et al., 2011; Ng & Petrick, 2019; Chitnis et al., 2021). Our lookahead strategy is most similar to the count-based approach considered by Lang et al. (2012), which in turn is related to the classic E^3 approach in the tabular setting (Kearns & Singh, 2002). Also relevant is work by Andersen & Konidaris (2017), who consider exploration in the context of learning symbolic state abstractions. These prior works typically consider finite action spaces, instead of the infinite action space we have here. Moreover, they operate in the model-based reinforcement learning (MBRL) setting (Eysenbach et al., 2019; Pathak et al., 2019; Colas et al., 2019), rather than the embodied active learning setting that we consider.

Active learning to ground natural language. At the intersection of natural language processing and robotics (Tellex et al., 2020), there is longstanding interest in learning to ground language. For example, Thomason et al. (2017) consider (non-embodied) active learning for visually grounding natural language descriptions of objects. Yang et al. (2018) study natural language query generation and propose an RL-based approach for selecting informative queries. Roesler & Nowé (2019) learn to ground natural language goals and learn policies for achieving those goals. We differ from these previous works in our focus on learning to plan and planning to learn. Given recent interest in using large language models (LLMs) for planning (Huang et al., 2022; Li et al., 2022; Ahn et al., 2022; Sharma et al., 2022),

a possible direction for future work would combine active learning for natural language grounding and LLM-based planning. However, recent studies suggest that classical AI planning techniques are still much stronger than LLM-based planners (Silver et al., 2022b; Valmeekam et al., 2023).

Embodied active learning. The challenge of interleaving action selection and active information gathering has been considered from many perspectives including *active reward learning* (Daniel et al., 2014; Schulze & Evans, 2018; Krueger et al., 2020), *active preference learning* (Sadigh et al., 2017; Biyik & Sadigh, 2018) and *interactive perception* (Bohg et al., 2017; Jayaraman & Grauman, 2018). We are especially influenced by Noseworthy et al. (2021), who actively learn to estimate the feasibility of abstract plans in TAMP. Also notable is recent work by Lamanna et al. (2023), who use known operators and AI planning methods to learn object properties through online exploration of a robotic environment. Finally, Kulick et al. (2013) consider active learning for relational symbol grounding, but in a non-sequential and discrete-action setting. Embodied active learning also shares certain facets with lifelong learning (Thrun, 1998; Abel et al., 2018) in that the agent improves incrementally and accumulates knowledge that helps it become better at learning in the future. Lifelong learning approaches that make use of hierarchy and abstraction for decision-making are most related to our efforts (Tessler et al., 2017; Wu et al., 2020; Lu et al., 2020). However, unlike lifelong learning, we do not address learning in non-stationary environments, nor do we attempt to learn incrementally (we retrain models from scratch).

7 Conclusion

In this paper, we proposed an embodied active learning framework for learning predicates useful for TAMP in continuous state and action spaces. Through experiments, we showed that the predicates are learned with strong sample efficiency in terms of both number of environment transitions and number of queries to the expert.

Limitations and Future Work. There are limitations of the present work and challenges for active predicate learning in general. In this work, we assumed an object-centric view of a fully-observed state; access to a deterministic simulator; and access to hybrid controllers. There is work on removing each of these assumptions with learning (Yuan et al., 2021; Wang et al., 2022; Chitnis et al., 2022; Silver et al., 2022a), but integration would be nontrivial. Assuming access to a deterministic simulator that exactly matches the environment is particularly unrealistic in real-world settings. In Appendix A.4, we present additional results with a noisy (but still known) simulator. We also used noise-free scripts to generate the initial demonstrations and expert responses. Since the agent is primarily learning from its own experience, we expect some robustness to noise in the initial demonstrations. To handle noise in the expert responses, we could model aleatoric uncertainty in addition to epistemic uncertainty, perhaps through the BALD objective (Houlsby et al., 2011; Noseworthy et al., 2021). In Appendix A.4, we also present additional experimental results where the expert gives noisy predicate labels.

For active predicate learning in general, one challenge is determining how frequently to relearn models. We relearned models after every episode, which led to strong sample complexity, but slowed down experiments overall (one run typically taking between 3 and 36 hours). Incremental learning approaches, especially for training the neural network predicate classifiers and samplers, could provide useful speedups (Castro et al., 2018; Ng & Petrick, 2019). Another issue is that the predicates given by the expert may be insufficient, or even unhelpful, for TAMP. Combining our approach with that of Silver et al. (2023) would help to address this issue since the agent could invent its own predicates, but we should also allow the agent to drop or modify expert-given predicates that it deems unhelpful. Finally, if learning predicates is one component of a larger learning-to-plan system, then predicate classifier entropy should not be the only driver of action selection: the agent's desire to learn better operators, samplers, controllers, state features, and so on, should also play a role in exploration.

8 Acknowledgements

We gratefully acknowledge support from NSF grant 2214177; from AFOSR grant FA9550-22-1-0249; from ONR MURI grant N00014-22-1-2740; from ARO grant W911NF-23-1-0034; from the MIT-IBM Watson Lab; from the MIT Quest for Intelligence; and from the Boston Dynamics Artificial Intelligence Institute. Tom is supported by a NSF Graduate Research Fellowship. We thank Jorge Mendez, Rohan Chitnis, Willie McClinton, and Leslie Kaelbling for helpful comments on an earlier draft.

REFERENCES

David Abel, David Hershkowitz, and Michael Littman. Near optimal behavior via approximate state abstraction. In *International Conference on Machine Learning*, pp. 2915–2923. PMLR, 2016.

David Abel, Dilip Arumugam, Lucas Lehnert, and Michael Littman. State abstractions for lifelong reinforcement learning. In *International Conference on Machine Learning*, pp. 10–19. PMLR, 2018.

Alper Ahmetoglu, M Yunus Seker, Justus Piater, Erhan Oztop, and Emre Ugur. Deepsym: Deep symbol generation and rule learning for planning from unsupervised robot interaction. *Journal of Artificial Intelligence Research*, 75: 709–745, 2022.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as I can, not as I say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Ahmed Akakzia, Cédric Colas, Pierre-Yves Oudeyer, Mohamed CHETOUANI, and Olivier Sigaud. Grounding language to autonomously-acquired skills via goal generation. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=chPj_I5KMHG.

Yusra Alkhazraji, Matthias Frorath, Markus Grützner, Malte Helmert, Thomas Liebetraut, Robert Mattmüller, Manuela Ortlieb, Jendrik Seipp, Tobias Springenberg, Philip Stahl, and Jan Wülfing. Pyperplan, 2020. URL https://doi.org/10.5281/zenodo.3700819.

Garrett Andersen and George Konidaris. Active exploration for learning symbolic representations. *Advances in neural information processing systems*, 30, 2017.

Erdem Biyik and Dorsa Sadigh. Batch active preference-based learning of reward functions. In *Conference on robot learning*, pp. 519–528. PMLR, 2018.

Jeannette Bohg, Karol Hausman, Bharath Sankaran, Oliver Brock, Danica Kragic, Stefan Schaal, and Gaurav S Sukhatme. Interactive perception: Leveraging action in perception and perception in action. *IEEE Transactions on Robotics*, 33(6):1273–1291, 2017.

Melissa Bowerman, Stephen C Levinson, and Stephen Levinson. *Language acquisition and conceptual development*. Cambridge University Press, 2001.

Marianella Casasola and Jui Bhagwat. Do novel words facilitate 18-month-olds' spatial categorization? *Child development*, 78(6):1818–1829, 2007.

Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 233–248, 2018.

Rohan Chitnis, Tom Silver, Josh Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Glib: Efficient exploration for relational model-based reinforcement learning via goal-literal babbling. In *AAAI*, 2021.

Rohan Chitnis, Tom Silver, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning neuro-symbolic relational transition models for bilevel planning. In *AAAI CLeaR Workshop*, 2022.

Cédric Colas, Pierre Fournier, Mohamed Chetouani, Olivier Sigaud, and Pierre-Yves Oudeyer. Curious: intrinsically motivated modular multi-goal reinforcement learning. In *International conference on machine learning*, pp. 1331–1340. PMLR, 2019.

Christian Daniel, Malte Viering, Jan Metz, Oliver Kroemer, and Jan Peters. Active reward learning. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014. doi: 10.15607/RSS.2014.X.031.

Carlos Diuk, Andre Cohen, and Michael L Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 240–247, 2008.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SJx63jRqFm.

Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning, 2020.

Carlos Guestrin, Daphne Koller, Chris Gearhart, and Neal Kanodia. Generalizing plans to new environments in relational mdps. In *Proceedings of the 18th international joint conference on Artificial intelligence*, pp. 1003–1010, 2003.

Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335–346, jun 1990. doi: 10.1016/0167-2789(90)90087-6. URL https://doi.org/10.1016.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

Jörg Hoffmann. Ff: The fast-forward planning system. *AI magazine*, 22(3):57–57, 2001.

Neil Houlsby, Ferenc Huszar, Zoubin Ghahramani, and Mate Lengyel. Bayesian active learning for classification and preference learning. In *NeurIPS Workshop on Bayesian optimization, experimental design and bandits: Theory and applications*, 2011.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning (ICML)*, 2022.

Dinesh Jayaraman and Kristen Grauman. Learning to look around: Intelligently exploring unseen environments for unknown tasks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1238–1247, 2018.

Nikolay Jetchev, Tobias Lang, and Marc Toussaint. Learning grounded relational symbols from continuous data for abstract reasoning, 2013.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49 (2-3):209–232, 2002.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

George Konidaris, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 16:215–289, 2018.

George Dimitri Konidaris and Andrew G Barto. Efficient skill learning using abstraction selection. In *IJCAI*, volume 9, pp. 1107–1112, 2009.

David Krueger, Jan Leike, Owain Evans, and John Salvatier. Active reinforcement learning: Observing rewards at a cost, 2020. URL https://arxiv.org/abs/2011.06709.

Johannes Kulick, Marc Toussaint, Tobias Lang, and Manuel Lopes. Active learning for teaching a robot grounded relational symbols. In *IJCAI*, pp. 1451–1457. Citeseer, 2013.

Leonardo Lamanna, Luciano Serafini, Mohamadreza Faridghasemnia, Alessandro Saffiotti, Alessandro Saetti, Alfonso Gerevini, and Paolo Traverso. Planning for learning object properties. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

Tobias Lang, Marc Toussaint, and Kristian Kersting. Exploration in relational domains for model-based reinforcement learning. *J. Mach. Learn. Res.*, 13(1):3725–3768, dec 2012. ISSN 1532-4435.

Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. In *AI&M*, 2006.

Shuang Li, Xavier Puig, Yilun Du, Clinton Wang, Ekin Akyurek, Antonio Torralba, Jacob Andreas, and Igor Mordatch. Pre-trained language models for interactive decision-making. *arXiv preprint arXiv:2202.01771*, 2022.

Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Reset-free lifelong learning with skill-space planning. *arXiv preprint arXiv:2012.03548*, 2020.

Toki Migimatsu and Jeannette Bohg. Grounding predicates through actions. *IEEE International Conference on Robotics and Automation (ICRA)*, 2022.

Jun Hao Alvin Ng and Ronald P. A. Petrick. Incremental learning of planning actions in model-based reinforcement learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 3195–3201. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/443. URL https://doi.org/10.24963/ijcai.2019/443.

Michael Noseworthy, Isaiah Brand, Caris Moses, Sebastian Castro, Leslie Kaelbling, Tomás Lozano-Pérez, and Nicholas Roy. Active learning of abstract plan feasibility. In *Robotics: Science and Systems XVII*. Robotics: Science and Systems Foundation, jul 2021. doi: 10.15607/rss.2021.xvii.043. URL https://doi.org/10.15607.

H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29:309–352, jul 2007. doi: 10.1613/jair.2113. URL https://doi.org/10.1613%2Fjair.2113.

Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5062–5071. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/pathak19a.html.

Christophe Rodrigues, Pierre Gérard, Céline Rouveirol, and Henry Soldano. Active learning of relational action models. In *International Conference on Inductive Logic Programming*, pp. 302–316. Springer, 2011.

Oliver Roesler and Ann Nowé. Action learning and grounding in simulated human–robot interactions. *The Knowledge Engineering Review*, 34:e13, 2019. doi: 10.1017/S0269888919000079.

Dorsa Sadigh, Anca D. Dragan, Shankar Sastry, and Sanjit A. Seshia. Active preference-based learning of reward functions. In *Robotics: Science and Systems*. Robotics: Science and Systems Foundation, 2017.

Sebastian Schulze and Owain Evans. Active Reinforcement Learning with Monte-Carlo Tree Search. *arXiv e-prints*, art. arXiv:1803.04926, March 2018.

Burr Settles. From theories to queries: Active learning in practice. In *Active learning and experimental design workshop in conjunction with AISTATS 2010*, pp. 1–18. JMLR Workshop and Conference Proceedings, 2011.

Pratyusha Sharma, Antonio Torralba, and Jacob Andreas. Skill induction and planning with latent language. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1713–1726, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.120. URL https://aclanthology.org/2022.acl-long.120.

Tom Silver, Rohan Chitnis, Joshua Tenenbaum, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Learning symbolic operators for task and motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

Tom Silver, Ashay Athalye, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning neuro-symbolic skills for bilevel planning. In *6th Annual Conference on Robot Learning*, 2022a. URL https://openreview.net/forum?id=OIaJRUo5UXy.

Tom Silver, Varun Hariprasad, Reece S Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Pddl planning with pretrained large language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022b.

Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomas Lozano-Perez, Leslie Pack Kaelbling, and Joshua Tenenbaum. Predicate invention for bilevel planning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. Robots that use language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:25–55, 2020.

Chen Tessler, Shahar Givony, Tom Zahavy, Daniel Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Jesse Thomason, Aishwarya Padmakumar, Jivko Sinapov, Justin Hart, Peter Stone, and Raymond J. Mooney. Opportunistic active learning for grounding natural language descriptions. In *Conference on Robot Learning*, pp. 67–76. PMLR, 2017.

Sebastian Thrun. *Lifelong Learning Algorithms*, pp. 181–209. Springer US, Boston, MA, 1998. ISBN 978-1-4615-5529-2. doi: 10.1007/978-1-4615-5529-2_8. URL https://doi.org/10.1007/978-1-4615-5529-2_8.

Karthik Valmeekam, Sarath Sreedharan, Matthew Marquez, Alberto Olmo, and Subbarao Kambhampati. On the planning abilities of large language models (a critical investigation with a proposed benchmark). *arXiv preprint arXiv:2302.06706*, 2023.

Thomas J Walsh. *Efficient learning of relational models for sequential decision making*. PhD thesis, Rutgers University-Graduate School-New Brunswick, 2010.

Chen Wang, Danfei Xu, and Li Fei-Fei. Generalizable task planning through representation pretraining. *IEEE Robotics and Automation Letters*, 7(3):8299–8306, 2022.

Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning compositional models of robot skills for task and motion planning. *The International Journal of Robotics Research*, 40(6-7):866–894, 2021. doi: 10.1177/02783649211004615. URL https://doi.org/10.1177/02783649211004615.

Bohan Wu, Jayesh K Gupta, and Mykel Kochenderfer. Model primitives for hierarchical lifelong reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 34:1–38, 2020.

Danfei Xu, Ajay Mandlekar, Roberto Martín-Martín, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Deep affordance foresight: Planning through what can be done in the future, 2020. URL https://arxiv.org/abs/2011.08424.

Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Visual curiosity: Learning to ask questions to learn visual recognition. *arXiv preprint arXiv:1810.00912*, 2018.

Wentao Yuan, Chris Paxton, Karthik Desingh, and Dieter Fox. Sornet: Spatial object-centric representations for sequential manipulation. In *5th Annual Conference on Robot Learning*, pp. 148–157. PMLR, 2021.

A Appendix

A.1 Additional Experimental Details

We now provide additional details for reproducing our experimental results.

A.2 Additional Details for Approaches

Learning. All neural networks are trained with the Adam optimizer (Kingma & Ba, 2014). For the classifier networks, each ensemble consists of 10 MLPs, and every MLP is trained for 100K epochs. Each MLP consists of two hidden layers, each of size 32, with ReLU activations. As mentioned in the main text, we use custom weight initialization to facilitate diversity in the ensemble. In particular, we initialize the MLP weights according to a $\mathcal{N}(0, 1)$ distribution. If the model fails to converge during training, we try reinitializing and retraining the model up to five times in total. Sampler neural network architecture and training is identical to that of Chitnis et al. (2022), except that we forgo training a discriminator and rejection sampling for simplicity. Operator learning is also identical to the prior work, except that we filter out any operators with less than 10 data points to facilitate efficient learning and planning, which was not needed in prior work because operators were learned from demonstrations instead of exploration data.

Planning. For planning in the evaluation tasks, all experiments use A* search with the LMCut heuristic for abstract planning, with the implementation for LMCut taken from Pyperplan (Alkhazraji et al., 2020). Following previous work, we consider up to $n_{\text{abstract}} = 8$ abstract plans per task and $n_{\text{samples}} = 10$ samples per step during refinement. Planning is timed out after 10 seconds.

Approach-specific details. The main entropy-based query policy uses a threshold of $\alpha = 0.05$, which we tuned to optimize performance in PickPlace1D. The GLIB baseline is GLIB-L1 from Chitnis et al. (2021). The Ask Randomly baseline asks about each possible ground atom with 0.03 probability, which we selected to approximately match the query rate of Main.

A.3 Additional Details for Environments

See Figure 5 for renderings of each environment. The details below for PickPlace1D and Blocks are modified with permission from Silver et al. (2023).

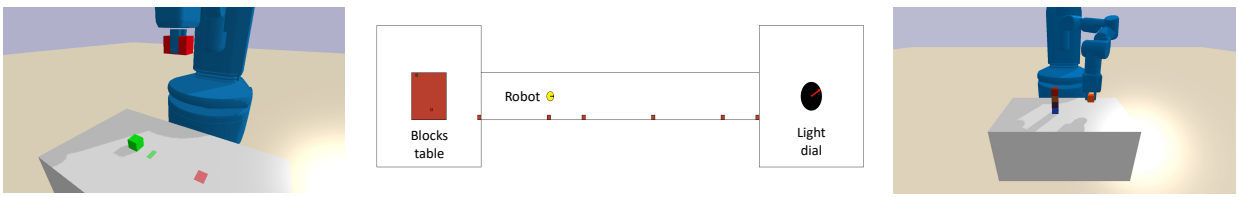

Figure 5: **Environments.** PickPlace1D and Blocks are from Chitnis et al. (2022), and Two Rooms is original to this work but loosely inspired by Konidaris & Barto (2009).

PickPlace1D. In this environment, a robot must pick blocks and place them onto target regions along a table surface. All pick and place poses are on a 1D line. The three object types are block, target, and robot. Blocks and targets each have two features for their pose and width. Robots have three features: a 2D pose and the (symmetric) value of the finger joint. The block widths are larger than the target widths, and the goal requires each block to be placed so that it completely covers the respective target region. The predicates to learn include `Covers(?block, ?target)`, `Holding(?robot, ?block)`, and `HandEmpty(?robot)`. There is only one controller, `PickPlace`, with no discrete parameters; its continuous parameter is a single real number denoting the location to perform either a pick or a place, depending on the current state of the robot's gripper. Each action updates the state of at most one block, based on whether any is in a small radius from the continuous parameter. During exploration and evaluation there are 2 blocks, 2 targets, and 1 robot. In each task, with 75% probability the robot starts out holding a random block; otherwise, both blocks start out on the table. This environment was established by Silver et al. (2021), but that work involved manually defined state abstractions, which we do not provide in this paper.

Two Rooms. In this environment, a robot in 3D starts out in a room with 3 blocks on a table. Beyond this room lies a hallway, and on the other end of the hallway is another room with a dial that controls a light. The three object types are block, robot, and dial. Blocks have five features: a 3D pose, a bit for whether it is currently grasped, and a bit for whether there the block is clear from above. The latter feature was added to make learning the `Clear(?block)` predicate possible, since we restrict our predicate classifiers to consider only the states of its arguments (Section 4). The robot has four features: a 2D position rotation for the base, and a (symmetric) value for the finger joints. The dial has three features: a 2D position and a level, where the level controls the light. The goal of an evaluation task is to build a certain block tower and turn the light on or off. There are six controllers: `Pick`, `Stack`, `PutOnTable`, `MoveTableToDial`, `TurnOnDial`, and `TurnOffDial`. The first three controllers are identical to Blocks; see below. The last three controllers are each parameterized by a robot, dial, and a 3D continuous change in pose. The predicates to learn include `On(?block1, ?block2)`, `OnTable(?block)`, `GripperOpen(?robot)`, `Holding(?robot, ?block)`, `Clear(?block)`, `NextToTable(?robot)`, `NextToDial(?robot, ?dial)`, `LightOn(?dial)`, and `LightOff(?dial)`. There are 3 blocks during exploration and in the evaluation tasks. This environment is original to this work.

Blocks. In this environment, a robot in 3D must interact with blocks on a table to assemble them into towers. This is a robotics adaptation of the blocks world domain in AI planning. The two object types are block and robot. Blocks have five features: an x/y/z pose, a bit for whether it is currently grasped, and a bit for whether there the block is clear from above. The latter feature was added to make learning the `Clear(?block)` predicate possible, since we restrict our predicate classifiers to consider only the states of its arguments (Section 4). Robots have four features: x/y/z end effector pose and a (symmetric) value for the finger joints. There are three controllers: `Pick`, `Stack`, and `PutOnTable`. `Pick` is parameterized by a robot and a block to pick up. `Stack` is parameterized by a robot and a block to stack the currently held one onto. `PutOnTable` is parameterized by a robot and a 2D place pose representing normalized coordinates on the table surface at which to place the currently held block. The predicates to learn include `On(?block1, ?block2)`, `OnTable(?block)`, `GripperOpen(?robot)`, `Holding(?robot, ?block)`, and `Clear(?block)`. During exploration there is 3 or 4 blocks, while evaluation tasks involve 5 or 6 blocks. This environment was established by Silver et al. (2021), but that work involved manually defined state abstractions, which we do not provide in this paper.

A.4 Additional Experimental Results

Here we present additional experimental findings.

Figure 6: **Failure analysis.** All results are averaged over 10 random seeds. Lines are means and shaded regions are 95% t-confidence intervals. See text for details.

Failure analysis. Figure 6 decomposes evaluation task failures into two categories: planning failures and execution failures. Planning failures occur when the agent cannot find a plan within the 10 second timeout and can be due to poor predicate interpretations, poor operators, or poor samplers. Execution failures occur when the agent finds a plan

but fails to reach the goal upon executing the plan in the environment. Since the agent has a perfect simulator of the deterministic environment, this type of failure only occurs when the agent has an incorrect interpretation of a goal predicate. For example, while planning with the simulator, the agent may find a plan that it believes achieves the goal `Covers(block1, target)`, but when it executes the plan, it actually fails to cover `block1` with `target` according to the expert's (ground-truth) interpretation of `Covers`.

Ensemble ablation. Figure 7 compares our main approach to an ablation that uses a single neural network for each learned predicate interpretation instead of an ensemble. Entropy is calculated on the basis of that single MLP neural network's predicted class probability. The stark difference between the ensemble and the single MLP confirms that an ensemble is vital for representing uncertainty in all environments.

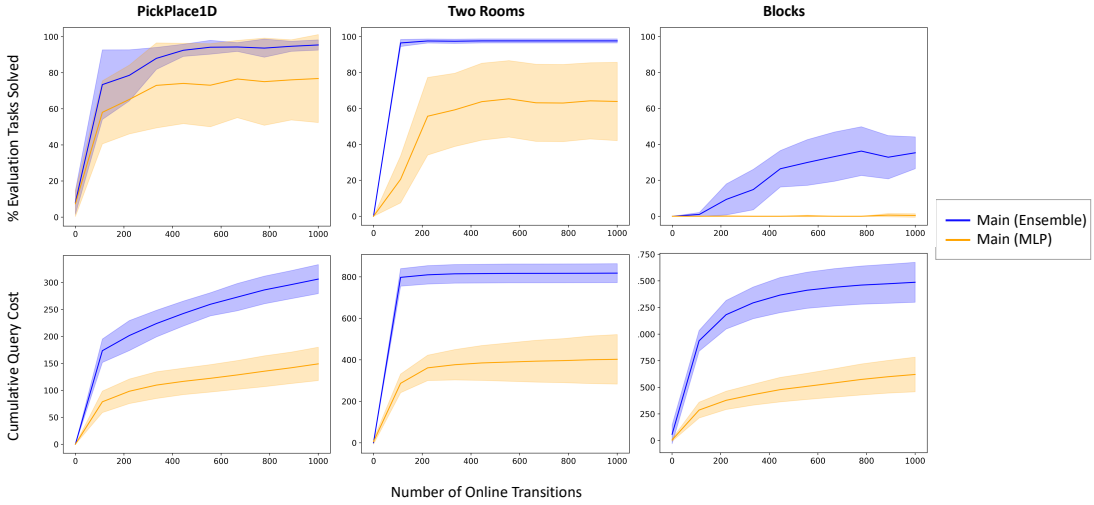

Figure 7: **Ensemble ablation.** All results are averaged over 10 random seeds. Lines are means and shaded regions are 95% t-confidence intervals. See text for details.

PickPlace1D Held-out Test Cases. In Figure 8, we show four illustrative states from PickPlace1D to demonstrate how the learned interpretation of the `Covers` predicate changes as the robot explores under the entropy-based query policy. Specifically, each set in the figure contains the ground atoms that result from the robot applying its interpretation of `Covers` to one of the four states at some point in time during learning. Ground-truth atoms are shown in the top row. Intuitively, the first (leftmost) and fourth (rightmost) states have the easiest classification problems for `Covers` because in the first, the green and red block are both clearly far from covering the target regions, and in the fourth, the red block completely overlaps the red target region. From just one episode after initialization, the agent correctly learns the classification for and becomes certain about these two states. The second and third states offer progressively more difficult classification problems, requiring more exploration. The agent classifies the second problem correctly by episode 3, and the third problem by episode 10.

Impact of Noisy Transitions. Figure 9 shows the impact of noisy environment transitions on our main approach. This experiment uses a version of the PickPlace1D environment such that when the robot takes an action to place a block on a target, the new position of the block is perturbed according to a $\mathcal{N}(0, 0.015)$ distribution. We find that this noticeably impacts the performance of our main approach, which is expected since the robot may encounter different noise during planning vs. during evaluation. Interestingly, the average query cost incurred is largely the same as before, suggesting that the robot's exploration and querying is similar to before, and since the robot fully observes its environment, that would mean the performance hit is largely attributed to planning.

Impact of Noisy Predicate Labels. Figure 10 shows the impact of noisy predicate labels on our main approach. In this experiment, any given ground atom label in an expert's response to a query is randomly inverted (made incorrect) with probability 0.05. We find that the noisy labels have a noticeable detrimental impact on performance: despite incurring much higher query cost, the robot solves fewer evaluation tasks, and there is greater variation in performance. We hypothesize that this is largely due to incorrectly labeled data points for the Covers predicate, since we have demonstrated that its classification boundary is tricky to learn (Figure 4).

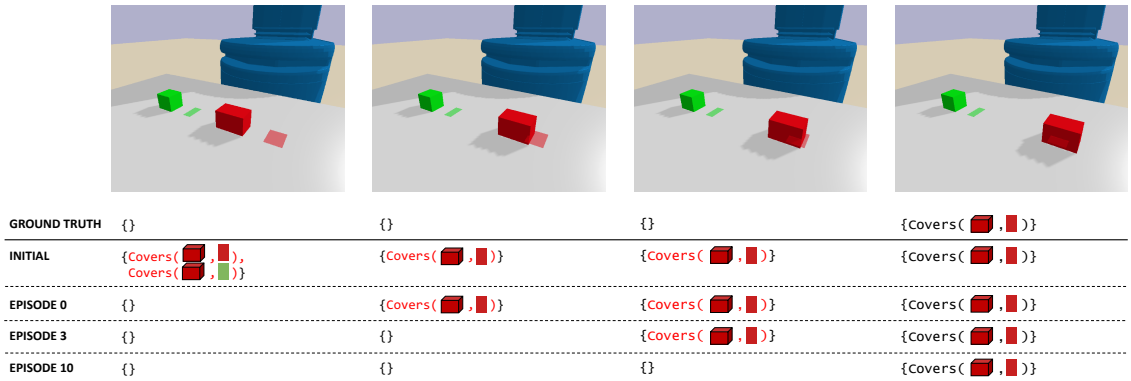

Figure 8: **PickPlace1D Held-out Test Cases.** From a single representative seed. Red text indicates a false interpretation compared to ground-truth while black indicates true. See text for details.

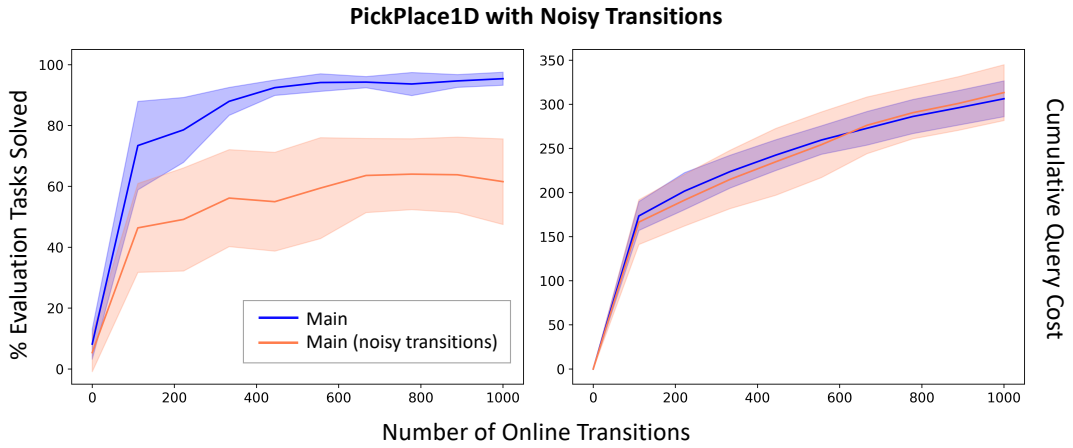

Figure 9: **PickPlace1D with Noisy Transitions.** All results are averaged over 10 random seeds. Lines are means and shaded regions are 95% t-confidence intervals. See text for details.

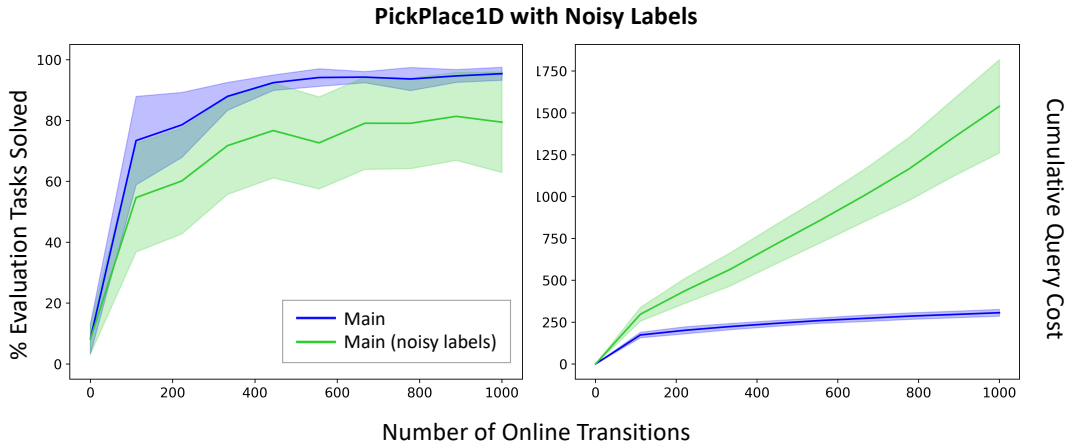

Figure 10: **PickPlace1D with Noisy Labels.** All results are averaged over 10 random seeds. Lines are means and shaded regions are 95% t-confidence intervals. See text for details.