Learning Neuro-Symbolic Skills for Bilevel Planning

Tom Silver, Ashay Athalye
Joshua B. Tenenbaum, Tomás Lozano-Pérez, Leslie Pack Kaelbling
MIT Computer Science and Artificial Intelligence Laboratory
{tslvr, ashay, jbt, tlp, lpk}@mit.edu

Abstract: Decision-making is challenging in robotics environments with continuous object-centric states, continuous actions, long horizons, and sparse feedback. Hierarchical approaches, such as task and motion planning (TAMP), address these challenges by decomposing decision-making into two or more levels of abstraction. In a setting where demonstrations and symbolic predicates are given, prior work has shown how to learn symbolic operators and neural samplers for TAMP with manually designed parameterized policies. Our main contribution is a method for learning parameterized polices in combination with operators and samplers. These components are packaged into modular *neuro-symbolic skills* and sequenced together with search-then-sample TAMP to solve new tasks. In experiments in four robotics domains, we show that our approach — bilevel planning with neuro-symbolic skills — can solve a wide range of tasks with varying initial states, goals, and objects, outperforming six baselines and ablations. Video: `https://youtu.be/PbFZP8rPuGg` Code: `https://tinyurl.com/skill-learning`

Keywords: Skill Learning, Neuro-Symbolic, Task and Motion Planning

1 Introduction

Decision-making in robotics environments with continuous state and action spaces is especially challenging when tasks have long horizons and goal-based objectives. A common strategy is to decompose decision-making into a high level ("what to do") and a low level ("how to do it") [1, 2, 3]. Seminal work by Konidaris et al. [4] considers such a hierarchy where symbolic AI planning [5] is used to sequence together temporally-extended continuous skills [6]. Their main contribution is a method for learning symbols from known skills. In this work, we study the inverse: learning skills from known symbols [7, 8, 9, 10, 11]. We are motivated by cases where it is easier to design [12, 13, 14, 15, 16] or learn [17, 18, 19, 20, 21] symbols than it is to design skills. In the long term, we envision a continually learning robot that uses symbols to learn skills and vice versa.

We consider symbols in the form of *predicates* — discrete relations between objects. A set of predicates induces a state abstraction [3, 22] of a continuous object-centric state space. For example, consider the *Stick Button* environment (Figure 1, third column), where a robot must press buttons either with its gripper, or by using a stick as a tool. Predicates include `Grasped` and `Pressed`, inducing an abstract state such as {`Grasped(robot,stick)`, `Pressed(button1)`, ...}. We leverage symbolic predicates to learn skills that generalize substantially from limited demonstration data. Concretely, we learn skills that achieve certain symbolic *effects* (e.g., `Grasped(robot, stick)`) from states where certain symbolic *preconditions* hold (e.g., `HandEmpty(robot)`).

Symbolic scaffolding for skill learning has two major benefits. First, by drawing on AI planning techniques, we can efficiently chain together long sequences of learned skills [4, 5, 10]. This benefit is key in our setting where planning in the low-level transition space is practically infeasible, even though a black-box transition function is deterministic and known — actions are continuous, solutions can exceed 100 actions, and there is no extrinsic feedback available before a task goal is reached. Second, because each skill acts in a confined region of the state space, learning is easier than it would be for a monolithic goal-conditioned policy (e.g., with behavioral cloning [23]).

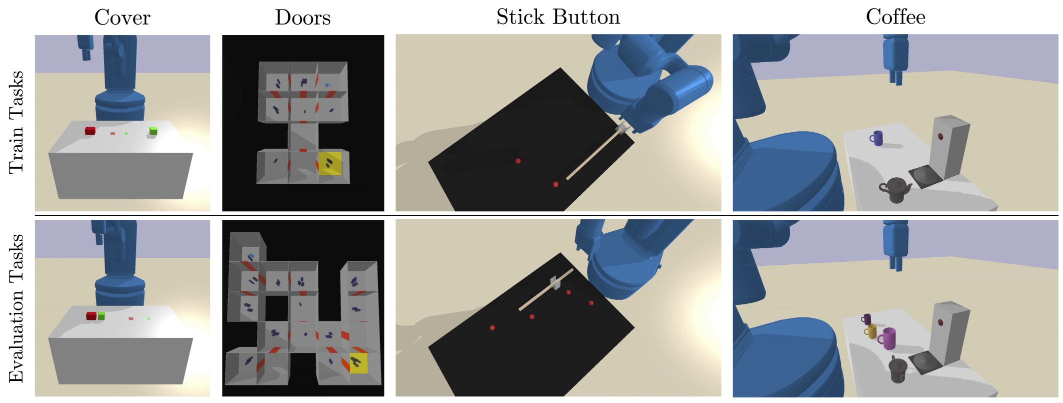

Figure 1: **Environments**. Top row: train task examples. Bottom row: evaluation task examples. See (§6) for descriptions of each environment.

While symbols offer useful structure for skill learning and planning, they also present challenges. In particular, symbolic state abstractions are often *lossy* [12, 13, 24], in that they discard information that may be important for decision-making, like the reachability of buttons or the relative stick grasp in Stick Button. This lossiness can be mitigated to some degree by inventing new predicates [4, 21], but in robotics environments, there are often kinematic and geometric constraints that are hard to perfectly abstract away [1, 25, 26]. These difficulties lead us to the following key desiderata.

Key Desideratum 1 (KD1). Because the abstraction is lossy, some environment states that correspond to the same symbolic state may be better than others in terms of completing a task. Thus, *a skill should be able to reach many different environment states ("subgoals") that correspond to the same abstract state.* For example, a skill that achieves `Grasped(robot, stick)` should not always lead to the same relative grasp: the grasp should be lower when buttons are farther away, and higher if collisions with the gray stick holder would prevent a bottom grasp. This desideratum implies that lookahead may be necessary, where the choice of skill early in the plan should be influenced by soft or hard constraints later in the plan. It also implies that we should not rely on the (abstract) subgoal condition in the framework of Konidaris et al. [4].

Key Desideratum 2 (KD2). When the state abstraction is lossy, it is not always possible to take actions in the environment that correspond to a particular abstract state sequence. Thus, *an agent should be able to consider multiple skill sequences that reach the same goal from the same initial abstract state.* For example, without a predicate that reflects button reachability, the agent may first plan to press all buttons without the stick, but if some buttons are unreachable, it will need the stick.

To address these two key desiderata, we propose an approach for learning and planning with *neuro-symbolic skills*. A neuro-symbolic skill consists of a symbolic operator [27], a neural subgoal-conditioned policy, and a neural subgoal sampler [28, 29]. The subgoals produced by the sampler and consumed by the policy correspond to different environment states that are consistent with the abstract effects of the operator, addressing KD1. The symbolic operators enable a bilevel planning algorithm [30, 31, 32], with AI planning in an outer loop and sampling in an inner loop. If sampling fails for an operator sequence, the outer loop continues to another sequence, addressing KD2. Skills are learned from demonstrations without any prior knowledge about the number of skills.

Main Contributions. Prior works by Silver et al. [12] and Chitnis et al. [13] propose a pipeline for learning operators and samplers for bilevel planning when *given* a set of parameterized policies[1]. While some policies may be human-designed and generally reusable, e.g., navigation policies that use motion planners, others are more domain-specific, such as pouring a pot of coffee or opening a door (Figure 1). Our main contribution is a method for learning these policies to complete the neuro-symbolic skills. We conduct experiments in four robotics environments (Figure 1) that require a wide range of behaviors, such as pouring, opening, pressing, picking, and placing. We find that skills learned from 100–250 demonstrations generalize to novel states, goals, and objects. Furthermore, our approach — bilevel planning with neuro-symbolic skills (BPNS) — outperforms six baselines and ablations, including learning graph neural network-based metacontrollers [15, 33].

[1]See (§C.3) for an elaborated discussion on the relationship between this work and [12, 13].

2 Problem Setting

We consider the problem of learning from demonstrations in deterministic, fully-observed environments with object-centric states, continuous actions, and a known transition function. An *environment* is characterized by a tuple $\langle \Lambda, \mathcal{U}, f, \Psi \rangle$. An object *type* $\lambda \in \Lambda$ has a name (e.g., `button`, `robot`) and a tuple of real-valued features (e.g., (`x`, `y`, `z`, `radius`, `color`, ...)) of dimension $\mathsf{dim}(\lambda)$. An *object* $o \in \mathcal{O}$ has a name (e.g., `button1`) and a type, denoted $\mathsf{type}(o) \in \Lambda$. A *state* $x \in \mathcal{X}$ is an assignment of objects to feature vectors, that is, $x(o) \in \mathbb{R}^{\mathsf{dim}(\mathsf{type}(o))}$ for $o \in \mathcal{O}$. The *action space* is denoted $\mathcal{U} \subseteq \mathbb{R}^m$. The *transition function* is a known, deterministic mapping from a state and action to a next state, denoted $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$. A *predicate* $\psi \in \Psi$ has a name (e.g., `Touching`) and a tuple of types (e.g., (`stick`, `button`)). A *ground atom* is a predicate and a mapping from its type tuple to objects (e.g., `Touching(stick, button1)`). A *lifted atom* instead has a mapping to typed variables, which are placeholders for objects (e.g., `Touching(?s, ?b)`). Predicates induce a state abstraction: $\mathsf{abstract}(x)$ denotes the set of ground atoms that hold true in x, with all others assumed false. We use $s \in \mathcal{S}$ to denote an abstract state, i.e., $\mathsf{abstract} : \mathcal{X} \to \mathcal{S}$.

An environment is associated with a *task distribution* $\mathcal{T}$, where each $T \in \mathcal{T}$ is characterized by a tuple $\langle \mathcal{O}, x_0, g, H \rangle$. The set of objects in the task is denoted $\mathcal{O}$. Importantly, objects vary between tasks. The initial state of the task is denoted $x_0 \in \mathcal{X}$. The goal, a set of ground atoms with predicates in Ψ and objects in $\mathcal{O}$, is denoted g. A goal represents a set of abstract states: for example, there are many possible abstract states where $g = \{$`Pressed(button2)`$\}$ holds. The task horizon, the limit for how many actions can be taken in a task, is denoted $H \in \mathbb{Z}^+$. A *solution* to a task is a sequence of actions that achieves the goal and does not exceed the task horizon, i.e. $\overline{u} = (u_1, \ldots, u_k)$ such that $k \leq H$, $g \subseteq \mathsf{abstract}(x_k)$, and $x_i = f(x_{i-1}, u_i)$ for $1 \leq i \leq k$.

We consider a standard learning setting where *train tasks* drawn from from $\mathcal{T}$ are available at training time, and held-out *evaluation tasks* drawn from $\mathcal{T}$ are used for evaluation. Our objective is to maximize the number of evaluation tasks solved within a wall-clock timeout. Each train task T is associated with one demonstration $\langle T, \overline{u}, \overline{x} \rangle$, where $\overline{u}$ is a solution for T and $\overline{x} = (x_0, \ldots, x_k)$ is the sequence of states visited. In experiments, tasks are randomly sampled, and demonstrations are generated with handcrafted environment-specific policies (but see (§C.4)).

3 Neuro-Symbolic Skills

How can we leverage the available demonstrations, predicates, and known transition model to maximize the number of evaluation tasks solved before timeout? We propose to learn and plan with neuro-symbolic skills. In this section, we define these skills formally; in (§4), we discuss planning; and in (§5), we address learning. See Figure 2 for a summary of the architecture.

A *skill* is a tuple $\phi = \langle \overline{v}, \omega, \pi, \sigma \rangle$ where $\overline{v}$ is a tuple of *arguments*; ω is an *operator*; π is a *subgoal-conditioned policy*; and σ is a *subgoal sampler*. A set of skills is denoted Φ. Arguments are variables with types in Λ and are shared by ω, π, and σ. An *operator* is a tuple $\omega = \langle \overline{v}, P, E^+, E^- \rangle$ where P, E^+, E^- are *preconditions*, *add effects*, and *delete effects* respectively, each a set of lifted atoms over the predicates Ψ and arguments $\overline{v}$. A *subgoal-conditioned policy* is a mapping $\pi : \mathcal{O}^{|\overline{v}|} \times \mathcal{X} \times \mathcal{X} \to \mathcal{U}$ that takes as input a tuple of objects for the arguments $\overline{v}$, the current state x, and a subgoal state x', and maps it to an action u. A *subgoal sampler* is a generator of elements from a distribution of subgoal states conditioned on object arguments and a current state, that is, $\sigma : \mathcal{O}^{|\overline{v}|} \times \mathcal{X} \to \Delta(\mathcal{X})$. A *ground skill* is a skill with objects that match its arguments, denoted by a tuple $\underline{\phi} = \langle \phi, \overline{o} \rangle$ where $\overline{o}$ is a tuple of objects in $\mathcal{O}$ with types matching $\overline{v}$. (We use underlines to denote grounded entities.) The corresponding *ground operator*, *ground policy*, and *ground sampler* are defined analogously,

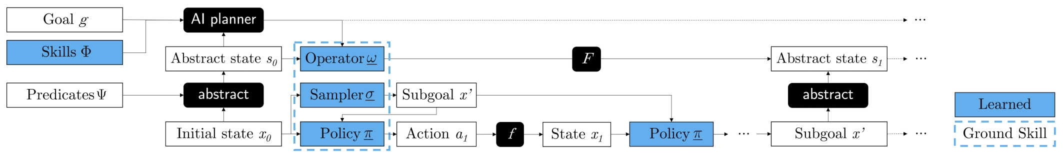

Figure 2: **Architecture**. Given learned symbolic operators, an AI planner is used to generate a candidate sequence of abstract states and skills. For each skill in the sequence, the learned sampler proposes a specific subgoal state in the next abstract state for the learned policy to pursue.

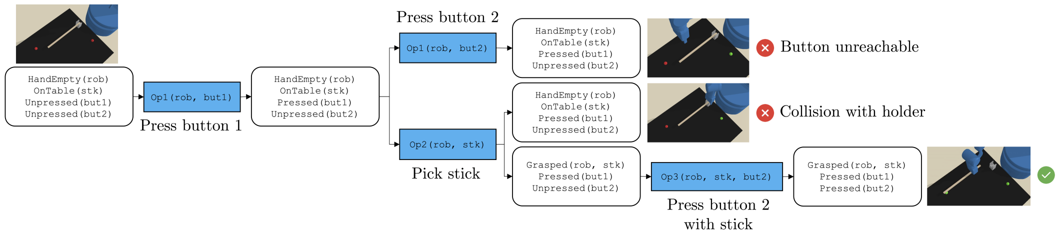

Figure 3: **Planning example**. First, an abstract plan that directly presses both buttons fails. Later, an abstract plan that uses the stick initially fails due to collisions, but succeeds on the next sample.

each associated with objects that match their typed variables, denoted $\underline{\omega} = \langle \underline{P}, \underline{E^+}, \underline{E^-} \rangle$, $\underline{\pi}$, and $\underline{\sigma}$ respectively. A set of ground skills $\underline{\Phi}$ induces an *abstract transition function* $F : \mathcal{S} \times \underline{\Phi} \to \mathcal{S}$ over the abstract state space through the add and delete effects of operators, given by $F(s, \underline{\phi}) = (s \setminus \underline{E^-}) \cup \underline{E^+}$, defined only when $\underline{P} \subseteq s$ (where $\setminus$ is set subtraction).

The contract between a skill's operator, policy, and sampler is the following: in states where the preconditions of a ground skill hold, i.e., $\underline{P} \subseteq \mathsf{abstract}(x)$, the sampler should generate a subgoal $x' \sim \underline{\sigma}(x)$ that satisfies the skill's effects, i.e. $F(\mathsf{abstract}(x), \underline{\phi}) = \mathsf{abstract}(x')$, and the actions proposed by the policy $\underline{\pi}(\cdot, x')$ should lead to x' from x. In practice, we *initiate* a ground skill only when the ground operator preconditions hold, and *terminate* at corresponding abstract successor state (cf. options [6]). Next, we show how to leverage this contract for efficient planning.

4 Bilevel Planning with Neuro-Symbolic Skills

Given skills and a new task, we search for a solution with bilevel planning. See Figure 3, Algorithm 1, and [13] for an extended description. Bilevel planning begins with an outer loop (Line 2), where skill operators are used to generate up to N_{abstract} *abstract plans*. An abstract plan for task $T = \langle \mathcal{O}, x_0, g, H \rangle$ is a sequence of ground skills that achieves the goal, i.e. $\overline{\phi} = (\underline{\phi}_1, \ldots, \underline{\phi}_\ell)$ such that $s_0 = \mathsf{abstract}(x_0)$, $g \subseteq s_\ell$, $\ell \leq H + 1$, and $s_i = F(s_{i-1}, \underline{\phi}_i)$ for $1 \leq i \leq \ell$.

```
  plan($T, \Phi, f$)
      // Params: $N_{\text{abstract}}, H_{\text{skill}}, N_{\text{samples}}$
1     $s_0 \leftarrow \mathsf{abstract}(x_0)$
      // Uses operators
2     for $\overline{\phi}$ in top-k($s_0$, $g$, $\Phi$, $N_{abstract}$) do
          // Uses samplers and policies
3         $\overline{u} \leftarrow$ refine($\overline{\phi}, T, \Phi, f, H_{\text{skill}}, N_{\text{samples}}$)
4         if $\overline{u}$ is not null then
              // Planning succeeded
5             return $\overline{u}$
      // Planning failed
6     return null
```

Algorithm 1: Bilevel planning.

Much work in the AI planning literature is devoted to efficiently generating these abstract plans, with most state-of-the-art techniques building on heuristic search. In experiments, we use A* search with the LMCut heuristic [34]. The AI planner details are not important for this work; we have also experimented with other planners and heuristics including GBFS, K* [35], FF [36], and various configurations of the Fast Downward planner [5]. Unlike typical AI planning, we continue the search after the first plan is found, generating a potentially infinite sequence of candidate plans (cf. top-k planning [35, 37, 38]).

Given a candidate abstract plan, we use the samplers and policies to perform a backtracking search over action sequences (Line 3). For each ground skill in the abstract plan, we use the ground sampler to produce a subgoal, and simulate the ground policy until either (1) the abstract transition is completed; or (2) a maximum skill horizon H_{skill} is reached. In case (1), we continue on to the next ground skill, or terminate if the goal is reached (Line 5), while in case (2) we backtrack up to N_{samples} times per step. If backtracking fails, we continue on to the next abstract plan [2].

[2]This planning strategy is not probabilistically complete, but see [32, 39] for complete variations.

5 Learning Neuro-Symbolic Skills from Demonstrations

We now address the problem of learning skills Φ from a dataset of demonstrations $\mathcal{D} = \{\langle T, \overline{u}, \overline{x} \rangle\}$. Recall that the data comprise raw sequences of states and actions for entire tasks. We preprocess the data in three steps: segmenting trajectories temporally, partitioning the segments into *skill datasets*, and lifting the objects within each skill dataset to variables. Then, for each skill dataset, we learn an operator, a subgoal-conditioned policy, and a subgoal sampler. We detail these steps below. This pipeline extends that of Chitnis et al. [13], who assume parameterized policies are given.

5.1 Data Preprocessing

Segmentation. For each demonstration $\langle T, \overline{u}, \overline{x} \rangle \in \mathcal{D}$, we identify a set of switch points $1 < i_1 < \cdots < i_\ell < |\overline{u}|$. Each pair of consecutive switch points (j, k) induces a *segment* $\gamma = \langle \overline{u}_{j:k}, \overline{x}_{j-1:k} \rangle$, where $\overline{u}_{j:k} = (u_j, u_{j+1}, \ldots, u_{k-1})$ is a subsequence of actions from $\overline{u}$ and $\overline{x}_{j-1:k}$ is defined analogously. Temporal segmentation is a much studied problem in the literature [40, 41, 42]. In this work, inspired by the notion of contact-based modes (e.g., [43]), we create a new switch point whenever there is a change in the truth-value of any contact-related ground atom (§A). Intuitively, using more (fewer) predicates here instead would lead to finer (coarser) temporal abstractions.

Partitioning. Next, we partition the set of all segments into a collection of *skill datasets*. We group segments into the same skill dataset if their abstract effects are equivalent up to object substitution. Formally, given a segment $\gamma = \langle \overline{u}_{j:k}, \overline{x}_{j-1:k} \rangle$, let $s = \mathtt{abstract}(x_{j-1})$ and $s' = \mathtt{abstract}(x_{k-1})$. The *effects* of the segment are $\langle e^+, e^- \rangle = \langle s' \setminus s, s \setminus s' \rangle$. The *affected object set* $\mathcal{O}^e$ comprises all objects that appear in either e^+ or e^-. Two segments γ_1, γ_2 with effects $\langle e_1^+, e_1^- \rangle$, $\langle e_2^+, e_2^- \rangle$ and affected objects $\mathcal{O}_1^e, \mathcal{O}_2^e$ are *equivalent* ($\gamma_1 \equiv \gamma_2$) if there exists an injective mapping $\delta : \mathcal{O}_1^e \to \mathcal{O}_2^e$ such that $\delta(e_1^+) = e_2^+$ and $\delta(e_1^-) = e_2^-$ and where each object in $\mathcal{O}_1^e$ shares the type of the object it is mapped to in $\mathcal{O}_2^e$. The relation $\equiv$ induces the partition: each partition contains equivalent segments.

Lifting. The segments in each skill dataset involve different objects. When we perform lifting, we replace all the objects in a segment with variables that have the objects' types. For example, in Stick Button, the `robot` may use the `stick` to press `button1` in some segment, and the `robot` may use the `stick` to press `button2` in another segment. After lifting, these objects would be replaced with `?robot`, `?stick`, and `?button`. Formally, for each skill dataset Φ_i, for an arbitrary segment $\gamma_0 \in \Phi_i$ with affected object set $\mathcal{O}_0^e$, we create one placeholder variable v for each object in $\mathcal{O}_0^e$. We order these variables arbitrarily into a tuple $\overline{v}$. Using the injective mappings δ described above, it is then possible to map the affected object set for any segment in Φ_i to the variables $\overline{v}$. In addition to aligning segments with different objects, variables define the *skill scope*: when computing abstract transitions, subgoals, and actions, the skills will reference only the object states for those variables. The final output of preprocessing is a set of lifted skill datasets, i.e., skill datasets ϕ_i, variables $\overline{v}$, and mappings from segment objects to variables that we will use in skill learning, described next.

5.2 Skill Learning

Operator Learning. Following previous work [13, 18, 21, 44], we use a simple linear-time approach to learn symbolic operators. We induce one operator from each lifted skill dataset Φ_i. The variables $\overline{v}$ constructed via lifting comprise the operator arguments. The operator add and delete effects follow immediately from lifting a segment's effects $\langle e^+, e^- \rangle$: we replace all objects with the corresponding variables in $\overline{v}$, to arrive at lifted atom sets $\langle E^+, E^- \rangle$. To compute preconditions, we use all common lifted atoms in the initial segment states. For a segment $\gamma \in \Phi_i$ with states $\overline{x}_{j-1:k}$, let $s_\gamma = \mathtt{abstract}(x_{j-1})$, and let $s_\gamma^{\overline{v}}$ denote the corresponding lifted atom set, with all affected objects in s_γ replaced with variables in $\overline{v}$, and with any ground atoms in s_γ involving objects not in $\overline{v}$ discarded. The preconditions P are then the intersection of lifted atom sets for each segment, denoted $P = \bigcap_{\gamma \in \Phi_i} s_\gamma^{\overline{v}}$, completing the operator $\omega = \langle \overline{v}, P, E^+, E^- \rangle$.

Policy Learning. Recall the responsibility of a ground policy $\underline{\pi}(x, x')$ is to output an action u that will lead the agent closer to the subgoal x' from the state x. At this point in the pipeline, for each skill, we have available a dataset of subgoals being achieved: the final state of each segment in the skill dataset is a subgoal, and the preceding states and actions demonstrate behavior towards that subgoal. We can therefore use these data for supervised learning of a subgoal-conditioned policy. First, we use the skill scope to convert the segment states into fixed-dimensional vectors by concatenating the features of the objects in the scope together with the subgoal. Formally, for each state x in each segment $\gamma \in \Phi_i$, we construct a vector $x^{\overline{v}} = x(v_1) \circ \cdots \circ x(v_n)$, where $\overline{v} = (v_1, \dots, v_n)$, $x(v_i) = x(o_i)$ for the corresponding o_i under lifting, and $\circ$ denotes vector concatenation. We then create a dataset of input-output pairs $(x^{\overline{v}}_{i-1} \circ x^{\overline{v}}_{k-1}, u_i)$ for $j \leq i < k$ in each segment with states $\overline{x}_{j-1:k}$, where $x^{\overline{v}}_{k-1}$ is the segment subgoal. We are now left with the problem of learning a policy from vector-based supervised data. In this work, we use behavioral cloning with fully-connected neural networks (see (§B) for details), but many other choices are possible[3]. At evaluation time, if the policy is grounded with objects $\overline{o} = (o_1, \dots, o_n)$, and queried in state x, then $x(o_1) \circ \cdots \circ x(o_n)$ is the state input to the neural network, while the subgoal input is generated by the ground sampler.

Sampler Learning. The role of the subgoal sampler is to propose different specific states, among the infinitely many consistent with the effects of the operator, for the policy to pursue. This flexibility is important for KD1 (§1). To learn samplers, we can largely reuse the datasets from policy learning: for each $(x \circ y, u)$ in the policy learning dataset, with y denoting the subgoal, we create a (x, y) input-output pair for sampler learning. Following [13], we learn two neural networks for each sampler. The first network outputs the mean and diagonal covariance of a Gaussian distribution. The second network is a binary classifier that takes a candidate (x, y) sampled from the Gaussian as input and accepts or rejects. These two networks are capable of modeling a richer class of distributions than the Gaussian alone, e.g., multimodal distributions. See (§B) for architecture and training details[4].

We make two modifications to the approach described above to promote generalization. First, we use the *relative* subgoal $x^{\overline{v}}_{k-1} - x^{\overline{v}}_{i}$ for each step i in the segment, rather than the absolute subgoal $x^{\overline{v}}_{k-1}$ (cf. goal relabelling [29, 46]). At evaluation time, we derive the absolute subgoal from the relative subgoal when the policy is first initialized, and update the relative subgoal after each transition. Second, we detect and remove subgoal dimensions that are static across all data (e.g., object mass).

Limitations. Our approach assumes that the given predicates comprise a useful state abstraction of the underlying state space with respect to the task distribution. With random or meaningless predicates, we would not expect to outperform non-symbolic baselines. However, previous work [12, 21] suggests that operator and sampler learning are robust to a limited degree to predicate ablation or addition; we expect the same for policy learning[5]. We assume that effective behavior can be determined from the skill scope alone and do not learn skills that handle a variable number of objects. Our method would create a separate skill for each unique skill dataset arising from affected object sets of different sizes, which might be preferred in some scenarios (picking up one, vs. ten, vs. two-hundred coins off the ground) but not others (tying multiple sticks together with a piece of string). Relational neural networks (e.g., GNNs) offer one possible path to address these two limitations. Through the lens of parameterized policy learning [47, 48, 49, 50], subgoals are just one form of parameterization among many possible alternatives. We also aspire to learn from *human* demonstrations on *real* robots; data efficiency results in the next section suggest that this is feasible.

6 Experimental Results

Our experiments address five questions about bilevel planning with neuro-symbolic skills (BPNS):

Q1. How many train tasks are required by BPNS to effectively solve held-out evaluation tasks?
Q2. Does BPNS generalize to unseen numbers of objects?
Q3. Can BPNS learn skills to complement existing general-purpose skills?
Q4. How important is the ability to sample multiple subgoals per abstract plan step (KD1)?
Q5. How important is the ability to generate multiple candidate abstract plans (KD2)?

Environments. We describe environments here at a high level, with details in (§A). The *Cover* environment was introduced by [12, 13, 21]. A robot must pick and place blocks to cover target regions on a 1D line. The robot can only initiate and release grasps in certain allowed regions. For example, if the robot grasps a block on the left side, it may not be able to place it to completely cover a target, depending on the allowed regions. Tasks have two blocks and two targets. There is random variation in the initial poses of the blocks, targets, allowed regions, and robot gripper.

[3]We also experimented briefly with implicit behavioral cloning [45] but did not notice any improvements.

[4]The policy and sampler for a skill are trained separately. In preliminary experiments, we observed that training them jointly sometimes performs better if training is stopped early, but is otherwise similar.

[5]See (§C.5) and (§C.6) for results with superfluous predicates and objects.

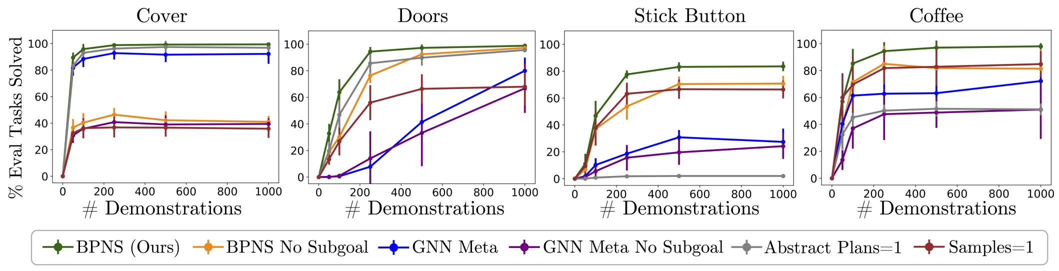

Figure 4: **Main results**. Evaluation task success rates versus number of demonstrations. Lines are means and error bars are standard deviation over 10 random seeds. GNN BC is excluded because its performance is consistently very poor, and its training is the most expensive; see the table below.

Doors is loosely inspired by the classic Four Rooms [6]. A robot must navigate to a target room while avoiding obstacles and opening doors. Doors have handles that require different rotations to open. In this environment, to address (**Q3**), a skill that moves the robot between configurations using a motion planner (BiRRT) is provided; door opening must be learned. Tasks have 4–25 rooms with random connectivity. The goals, obstacles, initial robot pose, and door parameters also vary.

The *Stick Button* environment is described in (§1). Train tasks have 1–2 buttons and evaluation tasks have 3–4 buttons. Tasks vary in the initial poses of the stick, holder, robot, and buttons.

In *Coffee*, a robot must move a pot of coffee to a hot plate, turn on the hot plate by pressing a button, and then move to pour the coffee into cups. The cups have different sizes and require different amounts of coffee. Depending on its orientation, the pot may need to be rotated before the handle can be grasped; the orientation is not reflected in the predicates. Overfilling a cup or pouring from too far results in a spill (failure). Train tasks have 1–2 cups and evaluation tasks have 2–3 cups. Tasks vary in the poses of the robot, pot, and cups, and in cup sizes. The goal is to fill all cups.

Approaches. We briefly describe the approaches we evaluate, with details in (§B).

- *BPNS*: Our main approach, bilevel planning with learned neuro-symbolic skills.
- *BPNS No Subgoal*: Our main approach, but with no samplers, and with subgoal-conditioned policies replaced with regular policies $\underline{\pi} : \mathcal{X} \to \mathcal{U}$. This approach is inspired by [7, 9, 51].
- *GNN Meta*: Instead of AI planning, this approach learns a graph neural network (GNN) meta-controller [9, 15, 33] that outputs a next skill based on the current state, abstract state, and goal.
- *GNN Meta No Subgoal*: Same as GNN Meta, but without samplers, and with regular policies. This baseline is the closest to existing methods in the AI planning literature [9, 10, 51, 52].
- *GNN BC*: Learns a monolithic goal-conditioned GNN policy by behavioral cloning (BC).
- *Samples=1*: An ablation of BPNS where $N_{\text{samples}} = 1$ during planning.
- *Abstract Plans=1*: An ablation of BPNS where $N_{\text{abstract}} = 1$ during planning.

All approaches are trained with the same demonstration data and use the same predicates. All approaches also use the transition function f, except for GNN BC, which is model-free.

Experimental Setup. For all environments and approaches, we use 50 evaluation tasks and 10 random seeds. The timeout for each evaluation task is 300 seconds and the horizon $H = 1000$. Experiments were run on Ubuntu 18.04 using 4 CPU cores of an Intel Xeon Platinum 8260 processor.

Results and Analysis. Figure 4 shows performance as a function of demonstration count. BPNS performs well in all environments after 100–250 demonstrations (**Q1**). Stick Button is the most challenging because many considered abstract plans are infeasible, which can lead to timeouts. The strong performance in Stick Button and Coffee, which have more objects than seen during training, also allows us to affirmatively resolve (**Q2**). Strong performance in Doors, where motion planners are provided, confirms that BPNS is capable of incorporating general-purpose skills (**Q3**).

The importance of generating multiple samples per abstract plan step (**Q4**) is reflected in the No Subgoal baselines and the Samples=1 ablation. The ablation performs worse than BPNS in all environments, while the No Subgoal baselines perform worse in all except Doors with 1000 demonstrations. We found evidence for two explanations in our analysis. First, skill policies may fail to terminate with certain parameters, but succeed with others. Second, even if a policy terminates, the agent may or may not be able to finish the rest of the task from that subgoal. These results confirm that KD1 (§1) is important for the strong performance of BPNS.

To analyze the importance of generating multiple candidate abstract plans (**Q5**), we can examine the GNN Meta baselines and the Abstract Plans=1 ablation. These approaches perform well in Cover and Doors, where the first abstract plan is usually refinable into a solution. In Coffee, the first abstract plan does not rotate the pot, and therefore fails when the handle is out of reach. See Figure 3 for an example of failure in Stick Button. These results confirm the importance of KD2 (§1).

We also compare BPNS with GNN BC, a purely model-free approach. The table on the right reports the mean (standard deviation) percent of evaluation tasks solved per environment after training with 1000 demonstrations. We also experimented with fewer demonstrations for GNN BC and saw consistently poor performance. The failure of GNN BC is unsurprising given the limited data and the high degree of task variability.

Environment	**BPNS (Ours)**	**GNN BC**
Cover	99.40 (0.64)	4.60 (1.62)
Doors	98.80 (0.80)	9.20 (4.05)
Stick Button	83.60 (1.78)	0.20 (0.30)
Coffee	98.00 (1.18)	23.80 (4.76)

In the appendix, we present additional results that investigate the planning time of BPNS (§C.1), the poor performance of GNN Meta (§C.2), comparison to [12, 13] (§C.3), learning from human demonstrations (§C.4), the impact of irrelevant predicates (§C.5) or objects (§C.6), the filtering of low-data skills (§C.7), comparison to oracle skills (§C.8), the structure of the learned operators (§C.9), and skill learning with invented predicates (§C.10).

7 Related Work

Our work contributes to a long line of research in skill learning for robotics, with prior works considering both reinforcement learning [53, 54, 55, 56, 57, 58, 59, 60, 61, 62] and learning from demonstrations [8, 40, 63, 64, 65, 66, 67, 68, 33]. In the context of hierarchical RL [28, 29, 69], it is possible to interpret our skill samplers as managers and policies as workers. Our work is also related to *parameterized* skill learning [47, 70, 48, 71, 49, 50] if we view subgoals as parameters. In the AI planning literature, prior works use a set of given planning operators to learn skills [7, 9, 10, 72, 73, 51, 52, 74]. Achterhold et al. [75] pursue a similar approach in a robotics setting. Recent work by Guan et al. [15] is closest to our contribution in that they use given operators to learn skills with latent parameters that can be used to reach a diverse set of terminal states [61], and then learn a metacontoller over the skills. The "GNN Meta" baseline is inspired by this work. Instead of operators, recent works have also used pretrained language models to generate abstract "instructions" and then learn language-conditioned skills [68, 76, 77, 78].

Our work can also be viewed as learning components of a task and motion planning (TAMP) system. In this line, other work has concentrated on learning operators [79, 80, 81, 82, 44, 12, 83], learning samplers [39, 84, 85, 86, 13, 87], or learning predicates [17, 88, 4, 89, 90, 91, 18, 19, 20, 92, 21]. Our operator and sampler learning algorithms are most directly related to that of Chitnis et al. [13], who assume given parameterized policies. Driess et al. [26] use given operators to learn energy-based controllers from images in the context of constraint-based TAMP [43]. Wang et al. [86] learn to sample continuously parameterized skills to accomplish symbolic operator effects, such as pouring at a certain angle to fill a cup. To the best of our knowledge, our work is the first to learn operators, samplers, and policies for search-then-sample TAMP [30, 31, 32].

8 Conclusion

In this work, we proposed bilevel planning with learned neuro-symbolic skills as a framework for decision-making in object-centric robotics environments. We confirmed experimentally that this approach leads to strong generalization and data efficiency. There are many interesting open questions for future work. (1) Can neuro-symbolic skills be learned from invented predicates? In (§C.10), we offer preliminary evidence suggesting the viability of this direction. (2) Can neuro-symbolic skills be learned with reinforcement learning? (3) Can latent object feature learning [16] be used to drive skill learning? Answering these questions will help us leverage hierarchical planning at scale.

Acknowledgements

We gratefully acknowledge support from NSF grant 1723381; from AFOSR grant FA9550-17-1-0165; from ONR grant N00014-18-1-2847; and from MIT-IBM Watson Lab. Tom is supported by an NSF Graduate Research Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors. We thank Rohan Chitnis, Nishanth Kumar, Willie B. McClinton, Aidan Curtis, Will Shen, and Anish Athalye for helpful comments on an earlier draft.

References

[1] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Perez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4, May 2021.

[2] P. Bercher, R. Alford, and D. Höller. A survey on hierarchical planning-one abstract idea, many concrete realizations. In *IJCAI*, pages 6267–6275, 2019.

[3] L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for MDPs. *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 4(5):9, 2006.

[4] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61: 215–289, 2018.

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

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

[7] M. R. Ryan. Using abstract models of behaviours to automatically generate reinforcement learning hierarchies. In *International Conference on Machine Learning (ICML)*, volume 2, pages 522–529. Citeseer, 2002.

[8] C. Paxton, F. Jonathan, M. Kobilarov, and G. D. Hager. Do what I want, not what I did: Imitation of skills by planning sequences of actions. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3778–3785. IEEE, 2016.

[9] D. Lyu, F. Yang, B. Liu, and S. Gustafson. SDRL: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2970–2977, 2019.

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

[11] Y. Wang, N. Figueroa, S. Li, A. Shah, and J. Shah. Temporal logic imitation: Learning plan-satisficing motion policies from demonstrations, 2022.

[12] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez. Learning symbolic operators for task and motion planning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3182–3189. IEEE, 2021.

[13] R. Chitnis, T. Silver, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling. Learning neuro-symbolic relational transition models for bilevel planning. *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[14] D. Driess, J.-S. Ha, and M. Toussaint. Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image. In *Robotics: Science and Systems 2020 (RSS 2020)*. RSS Foundation, 2020.

[15] L. Guan, S. Sreedharan, and S. Kambhampati. Leveraging approximate symbolic models for reinforcement learning via skill diversity. *arXiv preprint arXiv:2202.02886*, 2022.

[16] C. Wang, D. Xu, and L. Fei-Fei. Generalizable task planning through representation pretraining. *arXiv preprint arXiv:2205.07993*, 2022.

[17] N. Jetchev, T. Lang, and M. Toussaint. Learning grounded relational symbols from continuous data for abstract reasoning. In *Proceedings of the 2013 ICRA Workshop on Autonomous Learning*, 2013.

[18] M. Asai and C. Muise. Learning neural-symbolic descriptive planning models via cube-space priors: the voyage home (to strips). In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 2676–2682, 2021.

[19] A. Ahmetoglu, M. Y. Seker, J. Piater, E. Oztop, and E. Ugur. Deepsym: Deep symbol generation and rule learning from unsupervised continuous robot interaction for planning. *arXiv preprint arXiv:2012.02532*, 2020.

[20] E. Umili, E. Antonioni, F. Riccio, R. Capobianco, D. Nardi, and G. De Giacomo. Learning a symbolic planning domain through the interaction with continuous environments. *ICAPS PRL Workshop*, 2021.

[21] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Perez, L. P. Kaelbling, and J. Tenenbaum. Inventing relational state and action abstractions for effective and efficient bilevel planning. *arXiv preprint arXiv:2203.09634*, 2022.

[22] D. Abel, D. Hershkowitz, and M. Littman. Near optimal behavior via approximate state abstraction. In *International Conference on Machine Learning*, pages 2915–2923. PMLR, 2016.

[23] B. Fang, S. Jia, D. Guo, M. Xu, S. Wen, and F. Sun. Survey of imitation learning for robotic manipulation. *International Journal of Intelligent Robotics and Applications*, 3(4):362–369, 2019.

[24] B. Marthi, S. J. Russell, and J. A. Wolfe. Angelic semantics for high-level actions. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 232–239, 2007.

[25] M. Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[26] D. Driess, J.-S. Ha, R. Tedrake, and M. Toussaint. Learning geometric reasoning and control for long-horizon tasks from visual input. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14298–14305, 2021.

[27] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[28] P. Dayan and G. E. Hinton. Feudal reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 5, 1992.

[29] O. Nachum, S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[30] F. Gravot, S. Cambon, and R. Alami. aSyMov: a planner that deals with intricate symbolic and geometric problems. In *Robotics Research. The Eleventh International Symposium*. Springer, 2005.

[31] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *International Conference on Robotics and Automation (ICRA)*, pages 639–646. IEEE, 2014.

[32] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and Systems*, pages 1–6, 2016.

[33] Y. Zhu, P. Stone, and Y. Zhu. Bottom-up skill discovery from unsegmented demonstrations for long-horizon robot manipulation. *IEEE Robotics and Automation Letters*, 7(2):4126–4133, 2022.

[34] M. Helmert and C. Domshlak. Landmarks, critical paths and abstractions: what's the difference anyway? In *Nineteenth International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.

[35] M. Katz, S. Sohrabi, O. Udrea, and D. Winterer. A novel iterative approach to top-k planning. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS)*, 2018.

[36] J. Hoffmann. FF: The fast-forward planning system. *AI magazine*, 22(3):57–57, 2001.

[37] A. Riabov, S. Sohrabi, and O. Udrea. New algorithms for the top-k planning problem. In *Proceedings of the scheduling and planning applications workshop (spark) at the 24th international conference on automated planning and scheduling (ICAPS)*, pages 10–16, 2014.

[38] T. Ren, G. Chalvatzaki, and J. Peters. Extended tree search for robot task and motion planning. *arXiv preprint arXiv:2103.05456*, 2021.

[39] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel. Guided search for task and motion plans using learned heuristics. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 447–454. IEEE, 2016.

[40] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5239–5246, 2012.

[41] T. Kipf, Y. Li, H. Dai, V. Zambaldi, A. Sanchez-Gonzalez, E. Grefenstette, P. Kohli, and P. Battaglia. Compile: Compositional imitation learning and execution. In *International Conference on Machine Learning (ICML)*, pages 3418–3428. PMLR, 2019.

[42] T. Shankar, S. Tulsiani, L. Pinto, and A. Gupta. Discovering motor programs by recomposing demonstrations. In *International Conference on Learning Representations (ICLR)*, 2019.

[43] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Robotics: Science and Systems*, 2018.

[44] A. Arora, H. Fiorino, D. Pellier, M. Métivier, and S. Pesty. A review of learning planning action models. *The Knowledge Engineering Review*, 33, 2018.

[45] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR, 2022.

[46] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017.

[47] B. Da Silva, G. Konidaris, and A. Barto. Learning parameterized skills. *arXiv preprint arXiv:1206.6398*, 2012.

[48] J. Chang, N. Kumar, S. Hastings, A. Gokaslan, D. Romeres, D. Jha, D. Nikovski, G. Konidaris, and S. Tellex. Learning deep parameterized skills from demonstration for re-targetable visuomotor control. *arXiv preprint arXiv:1910.10628*, 2019.

[49] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine. Parrot: Data-driven behavioral priors for reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2021.

[50] A. Ajay, A. Kumar, P. Agrawal, S. Levine, and O. Nachum. OPAL: Offline primitive discovery for accelerating offline reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2021.

[51] L. Illanes, X. Yan, R. T. Icarte, and S. A. McIlraith. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 30, pages 540–550, 2020.

[52] V. Sarathy, D. Kasenberg, S. Goel, J. Sinapov, and M. Scheutz. SPOTTER: Extending symbolic planning operators through targeted reinforcement learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1118–1126, 2021.

[53] A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. *International Conference on Machine Learning (ICML)*, 2001.

[54] A. Bagaria and G. Konidaris. Option discovery using deep skill chaining. In *International Conference on Learning Representations (ICLR)*, 2019.

[55] G. Konidaris and A. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2009.

[56] E. Brunskill and L. Li. Pac-inspired option discovery in lifelong reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 316–324, 2014.

[57] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[58] K. Gregor, D. J. Rezende, and D. Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

[59] V. Thomas, J. Pondard, E. Bengio, M. Sarfati, P. Beaudoin, M.-J. Meurs, J. Pineau, D. Precup, and Y. Bengio. Independently controllable features. *arXiv preprint arXiv:1708.01289*, 2017.

[60] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman. Meta learning shared hierarchies. *International Conference on Learning Representations (ICLR)*, 2018.

[61] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. *International Conference on Learning Representations (ICLR)*, 2019.

[62] M. Riemer, M. Liu, and G. Tesauro. Learning abstract options. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[63] P. Ranchod, B. Rosman, and G. Konidaris. Nonparametric bayesian reward segmentation for skill discovery using inverse reinforcement learning. In *IROS*, pages 471–477. IEEE, 2015.

[64] C. Daniel, H. Van Hoof, J. Peters, and G. Neumann. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104(2-3):337–357, 2016.

[65] R. Fox, S. Krishnan, I. Stoica, and K. Goldberg. Multi-level discovery of deep options. In *CoRL*, 2017.

[66] A. Mandlekar, D. Xu, R. Martín-Martín, S. Savarese, and L. Fei-Fei. GTI: learning to generalize across long-horizon tasks from human demonstrations. In *Robotics: Science and Systems XVI*, 2020.

[67] D. Tanneberg, K. Ploeger, E. Rueckert, and J. Peters. Skid raw: Skill discovery from raw trajectories. *IEEE Robotics and Automation Letters*, 6(3):4696–4703, 2021.

[68] P. Sharma, A. Torralba, and J. Andreas. Skill induction and planning with latent language. *arXiv preprint arXiv:2110.01517*, 2021.

[69] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *International Conference on Machine Learning (ICML)*, 2017.

[70] B. C. Da Silva, G. Baldassarre, G. Konidaris, and A. Barto. Learning parameterized motor skills on a humanoid robot. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5239–5244. IEEE, 2014.

[71] K. Pertsch, Y. Lee, and J. J. Lim. Accelerating reinforcement learning with learned skill priors. *arXiv preprint arXiv:2010.11944*, 2020.

[72] M. Grounds and D. Kudenko. Combining reinforcement learning with symbolic planning. In *Adaptive Agents and Multi-Agent Systems (AAMAS)*, pages 75–86. Springer, 2005.

[73] F. Yang, D. Lyu, B. Liu, and S. Gustafson. PEORL: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. *arXiv preprint arXiv:1804.07779*, 2018.

[74] M. Jin, Z. Ma, K. Jin, H. H. Zhuo, C. Chen, and C. Yu. Creativity of AI: Automatic symbolic option discovery for facilitating deep reinforcement learning. *arXiv preprint arXiv:2112.09836*, 2021.

[75] J. Achterhold, M. Krimmel, and J. Stueckler. Learning temporally extended skills in continuous domains as symbolic actions for planning. *arXiv preprint arXiv:2207.05018*, 2022.

[76] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, and M. Yan. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022.

[77] S. Li, X. Puig, Y. Du, C. Wang, E. Akyurek, A. Torralba, J. Andreas, and I. Mordatch. Pretrained language models for interactive decision-making. *arXiv preprint arXiv:2202.01771*, 2022.

[78] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022.

[79] H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29:309–352, 2007.

[80] H. H. Zhuo, Q. Yang, D. H. Hu, and L. Li. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, 174(18):1540–1569, 2010.

[81] Q. Yang, K. Wu, and Y. Jiang. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171(2-3):107–143, 2007.

[82] H. H. Zhuo, T. Nguyen, and S. Kambhampati. Refining incomplete planning domain models through plan traces. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[83] P. Verma, S. R. Marpally, and S. Srivastava. Discovering user-interpretable capabilities of black-box planning agents. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2022.

[84] A. Mandalika, S. Choudhury, O. Salzman, and S. Srinivasa. Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 745–753, 2019.

[85] R. Chitnis, L. P. Kaelbling, and T. Lozano-Pérez. Learning quickly to plan quickly using modular meta-learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7865–7871. IEEE, 2019.

[86] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. 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.

[87] J. Ortiz-Haro, J.-S. Ha, D. Driess, and M. Toussaint. Structured deep generative models for sampling on constraint manifolds in sequential manipulation. In *Conference on Robot Learning*, pages 213–223. PMLR, 2022.

[88] E. Ugur and J. Piater. Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2627–2633. IEEE, 2015.

[89] B. Ames, A. Thackston, and G. Konidaris. Learning symbolic representations for planning with parameterized skills. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 526–533. IEEE, 2018.

[90] J. Loula, T. Silver, K. R. Allen, and J. Tenenbaum. Discovering a symbolic planning language from continuous experience. In *CogSci*, page 2193, 2019.

[91] S. James, B. Rosman, and G. Konidaris. Learning portable representations for high-level planning. In *International Conference on Machine Learning (ICML)*, pages 4682–4691. PMLR, 2020.

[92] A. Curtis, T. Silver, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling. Discovering state and action abstractions for generalized task and motion planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.