Learning Neuro-Symbolic Skills for Bilevel Planning

by

Ashay Athalye

S.B. Electrical Engineering and Computer Science and Economics
Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2023

Authored by:	Ashay Athalye Department of Electrical Engineering and Computer Science August 25, 2023
Certified by:	Leslie P. Kaelbling Panasonic Professor of Computer Science and Engineering, Thesis Supervisor
Certified by:	Tom Silver Ph.D. Student, Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

Learning Neuro-Symbolic Skills for Bilevel Planning

by

Ashay Athalye

Submitted to the Department of Electrical Engineering and Computer Science
on August 25, 2023 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

It is challenging for robots to solve tasks in environments with continuous state and action spaces, long horizons, and sparse feedback. Hierarchical approaches such as task and motion planning (TAMP) address this challenge, enabling efficient problem solving by decomposing decision-making into two or more levels of abstraction. In a setting where expert demonstrations, symbolic predicates for state abstraction, and manually designed parameterized policies are given, prior work has shown how to learn symbolic operators and neural samplers for TAMP. But Manually designing parameterized policies can be difficult and impractical, so we would instead like our agent to learn them. In this work, we develop a method for learning parameterized polices in combination with operators and samplers from demonstrations. 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, objects, and goals, outperforming six baselines and ablations.

Thesis supervisor: Leslie P. Kaelbling

Title: Panasonic Professor of Computer Science and Engineering

Thesis supervisor: Tom Silver

Title: Ph.D. Student

Acknowledgments

First, I want to thank Tom. Tom has been the best research mentor I've had. I started the M.Eng. program unsure if research was what I wanted to do, unsure if it was right for me, unsure if I was good at it. After my time with Tom and with the Learning and Intelligent Systems group, I know that I want to do research, I know that it's right for me, and I know that I can be good at it. I'm now confident that I love research.

Tom is so supportive, smart, hardworking, kind, funny, motivating, and inspiring. Under his mentorship, I have had everything I need to succeed. And that's the best feeling you can have when doing anything – that the only limit to what you can reach is how much effort you put in, that there's nothing else holding you back. Tom's mentorship makes you feel like you can accomplish anything.

Second, I want to thank members of our lab, starting with faculty members Leslie, Tomàs, and Josh, frequent collaborators Nishanth and Willie, and everyone else, for their wisdom, comments, and support.

Third, I want to thank Leslie for giving me the opportunity to do research in the lab. I'm so grateful to have had this opportunity. This experience has changed my life in many ways, including fostering a deep love and appreciation for research.

Finally, I thank my Mom, my Dad, and my Brother, Anish, for their infinite support and love. I couldn't do anything without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

It is challenging for robots to solve tasks in environments with continuous state and action spaces, long horizons, and sparse feedback. Solving such tasks with a goal-conditioned monolithic reactive policy learned with behavioral cloning or reinforcement learning typically exhibits sample inefficiency and poor generalization because the policy needs to incorporate both what subtasks to do in order to achieve the goal and how to control the robot to execute the desired motions for each subtask. For example, cooking a dish from scratch in a kitchen environment may require a long sequence of opening cabinets, moving containers, cutting vegetables, mixing ingredients, and using appliances, with each of these steps requiring many actions. A common strategy to enable efficient problem solving in this setting is to decompose decision-making into a high level ("what to do") and a low level ("how to do it") [3–5]. Seminal work by Konidaris et al. [6] considers such a hierarchy where symbolic AI planning [7] is used to sequence together temporally-extended continuous skills [8]. Their main contribution is a method for learning symbols from known skills. In this work, we study the inverse: learning skills from known symbols [9–13]. We are motivated by cases where it is easier to design [1, 2, 14–16] or learn [17–21] 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 [5, 22] of a continuous object-centric state space. For example, consider the *Stick Button* environment (Figure 1.1, 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 [6, 7, 12]. 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]).

While symbols offer useful structure for skill learning and planning, they also present challenges. In particular, symbolic state abstractions are usually and intentionally *lossy* [1, 2, 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 [6, 21], but in robotics environments, there are often kinematic and geometric constraints that are hard to perfectly abstract away [3, 25–27]. 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

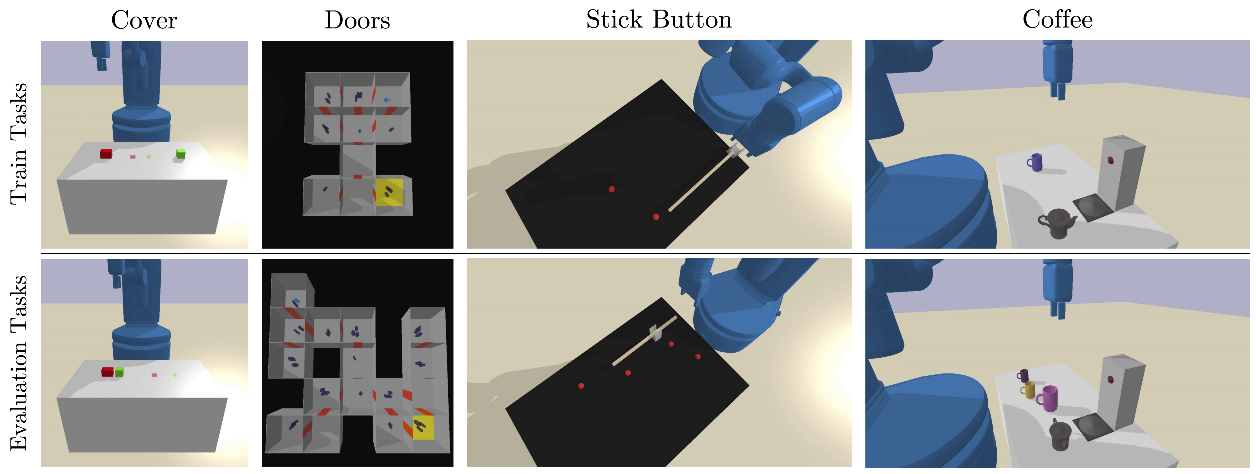


Figure 1.1: **Environments**. Top row: train task examples. Bottom row: evaluation task examples. See (§6) for descriptions of each environment.

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.

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 (i.e. the downward refinement property does not hold [28]). 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 [29], a neural subgoal-conditioned policy, and a neural subgoal sampler [30, 31]. 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 [32–34], 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[1]**.** Prior works by Silver et al. [1] and Chitnis et al. [2] propose a pipeline for learning operators and samplers for bilevel planning when *given* a set of parameterized policies[2]. 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 1.1). Our main contribution is a method for learning these policies to complete the neuro-symbolic skills. We conduct experiments in four robotics environments (Figure 1.1) 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, objects, and goals. Furthermore, our approach — bilevel planning with neuro-symbolic skills (BPNS) — outperforms six baselines and ablations, including learning graph neural network-based metacontrollers [15, 36].

[1]Note that the work presented in this thesis has been previously published in [35].
[2]See (§C.3) for an elaborated discussion on the relationship between this work and [1, 2].

Chapter 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`, $\ldots$)) 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 $g = \{\texttt{Pressed(button2)}\}$ 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 (see (§C.5) for an additional experiment with human demonstrations).

Chapter 3

Neuro-Symbolic Skills

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 3.1 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 $\pi : \mathcal{O}^{|\overline{v}|} \times \mathcal{X} \times \mathcal{X} \rightarrow \mathcal{U}$ 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.

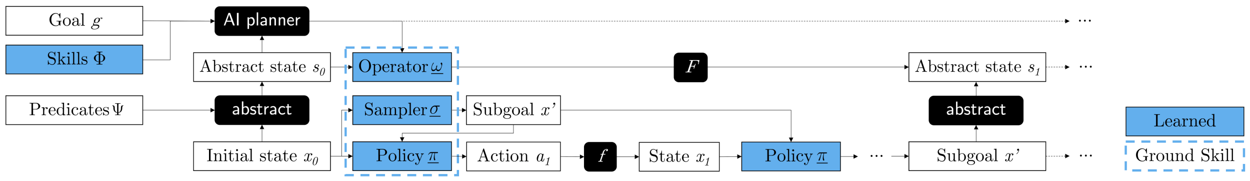


Figure 3.1: **Architecture**. Given learned symbolic operators, an AI planner is used to generate a candidate sequence of abstract states and skills. For each skill in the sequence, the learned sampler proposes a specific subgoal state in the next abstract state for the learned policy to pursue.

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, 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).

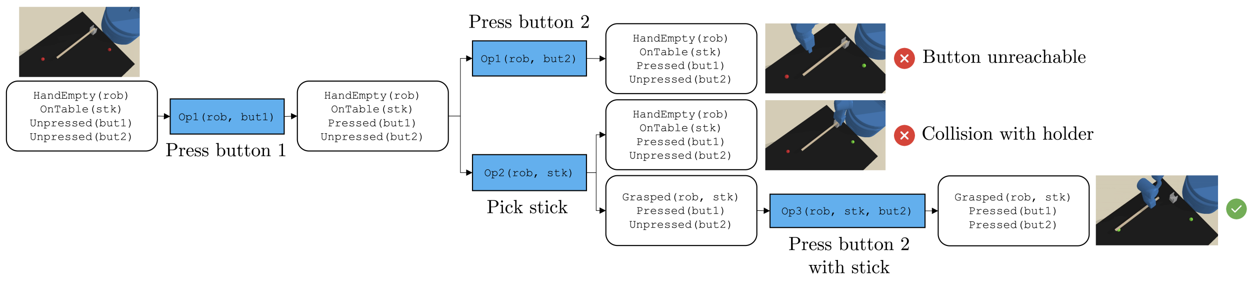
A skill's operator, policy, and sampler work together in the following way: in states where the preconditions of a ground skill's operator hold, i.e., $\underline{P} \subseteq \mathsf{abstract}(x)$, the sampler should generate a subgoal $x' \sim \underline{\sigma}(x)$ that satisfies the effects of the skill's operator, 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 [8]). Next, we show how to leverage neuro-symbolic skills for efficient planning.

Chapter 4

Bilevel Planning with Neuro-Symbolic Skills

Given skills and a new task, we search for a solution with bilevel planning. See Figure 4.1, Algorithm 1, and [2] 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. $\overline{\underline{\phi}} = (\underline{\phi}_1, \ldots, \underline{\phi}_\ell)$ 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 $1 \leq i \leq \ell$.

A lot of work in the artificial intelligence planning literature is devoted to efficiently generating these abstract plans, with most state-of-the-art techniques building on heuristic search. In our experiments, we use A* search with the LMCut heuristic [37]. The details of the planner are not important for this work; we have also experimented with other planners and heuristics including GBFS, K* [38], FF [39], and various configurations of the Fast Downward planner [7]. 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 [38, 40, 41]).



HandEmpty(rob) OnTable(stk) Unpressed(but1) Unpressed(but2)
Op1(rob, but1)
Press button 1
HandEmpty(rob) OnTable(stk) Pressed(but1) Unpressed(but2)
Press button 2
Op1(rob, but2)
Op2(rob, stk)
Pick stick
HandEmpty(rob) OnTable(stk) Pressed(but1) Unpressed(but2)
Button unreachable
HandEmpty(rob) OnTable(stk) Pressed(but1) Unpressed(but2)
Collision with holder
Grasped(rob, stk) Pressed(but1) Unpressed(but2)
Op3(rob, stk, but2)
Press button 2 with stick
Grasped(rob, stk) Pressed(but1) Pressed(but2)

Figure 4.1: **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.

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

Algorithm 1: Bilevel planning.

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. Note that this planning strategy is not probabilistically complete, but see [34, 42] for complete variations.

Chapter 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. This pipeline extends that of Chitnis et al. [2], who assume parameterized policies are given. Before detailing these steps, we discuss structural biases that can be leveraged in our environments.

5.1 Low-Dimensional State Changes and Skill Scopes

Robotics environments with object-centric state spaces often exhibit low-dimensional state changes in their transition space, especially in demonstrations. For example, in a Stick Button environment with a large number of buttons, at most one button changes state at any given time. We make use of this observation during skill learning in two ways. First, rather than dealing with absolute changes in states, we model *changes* in states. This commitment has already manifested in the effects of symbolic operators, and it arises again below in the subgoal interface between samplers and policies. Second, instead of using the full object-centric state as input to skills, we use an automatically-derived subset of the objects, which we refer to as the *scope* of the skill. For example, the ground skill policy for pressing a particular button with a stick will be a function of the state of the robot, the stick, and that button. This second commitment is strong because in a general case, optimal behavior may depend on the full state. We postulate, and confirm in experiments, that a wide variety of interesting skills can be learned with a small and restricted object scope, and leave open the possibility for extensions where the full state is incorporated.

5.2 Data Preprocessing

5.2.1 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 [43–45]. In this work, inspired by the notion of contact-based modes (e.g., [46]), we create a new switch point whenever there is a change in the truth-value of any contact-related ground atom (see (§A) for contact-related predicates in each environment). Intuitively, using more (fewer) predicates here instead would lead to finer (coarser) temporal abstractions.

5.2.2 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 $s = \mathsf{abstract}(x_{j-1})$ and $s' = \mathsf{abstract}(x_{k-1})$. 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 $\delta : \mathcal{O}_1^e \rightarrow \mathcal{O}_2^e$ 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.

5.2.3 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.

5.3 Skill Learning

5.3.1 Operator Learning

Following previous work [2, 18, 21, 47], 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 $s_\gamma = \mathsf{abstract}(x_{j-1})$, 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 $\omega = \langle \overline{v}, P, E^+, E^- \rangle$.

5.3.2 Policy Learning

Recall that 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 $\overline{v} = (v_1, \ldots, v_n)$, $x(v_i) = x(o_i)$ 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[1]. At evaluation time, if the policy is grounded with objects $\overline{o} = (o_1, \ldots, o_n)$, 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.

5.3.3 Sampler Learning

The role of the subgoal sampler is to propose different specific states for the policy to pursue, among the infinitely many consistent with the effects of the operator. 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 [2], 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[2].

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 [31, 49]). At evaluation time, we derive the absolute subgoal from the relative subgoal when the policy is first initialized, and update

[1]We also experimented briefly with implicit behavioral cloning [48] but did not notice any improvements.

[2]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.

the relative subgoal after each transition. Second, we detect and remove subgoal dimensions that are static across all data (e.g., object mass).

5.4 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 [1, 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[3]. 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. The restriction of a skill's inputs to its *scope* relies on the assumption that good behavior can be determined from a low-dimensional subspace of the state. This assumption would be violated in situations where the agent needs to attend to a large number of objects to make a decision, e.g., in a domain that requires object counting. 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. In learning a *deterministic* subgoal-conditioned policy, we are assuming that the agent does not to consider multiple paths from the same state to the same subgoal. While it is technically true that the ability to complete a task starting at a subgoal is independent from how the agent arrived at that subgoal, stochasticity may still be helpful in the same way that probabilistic motion planning algorithms benefit from stochasticity in deterministic environments. Through the lens of parameterized policy learning [50–53], subgoals are just one form of parameterization among many possible alternatives. We also

[3]See (§C.6) and (§C.7) for results with superfluous predicates and objects.

aspire to learn from *human* demonstrations on *real* robots; data efficiency results in the next section and preliminary experiments in (§C.5) suggest that this is feasible.

Chapter 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)?

6.1 Environments

We describe environments here at a high level, with details in (§A). The *Cover* environment was introduced by [1, 2, 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.

Doors is loosely inspired by the classic Four Rooms [8]. 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.

6.2 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 [9, 11, 54].

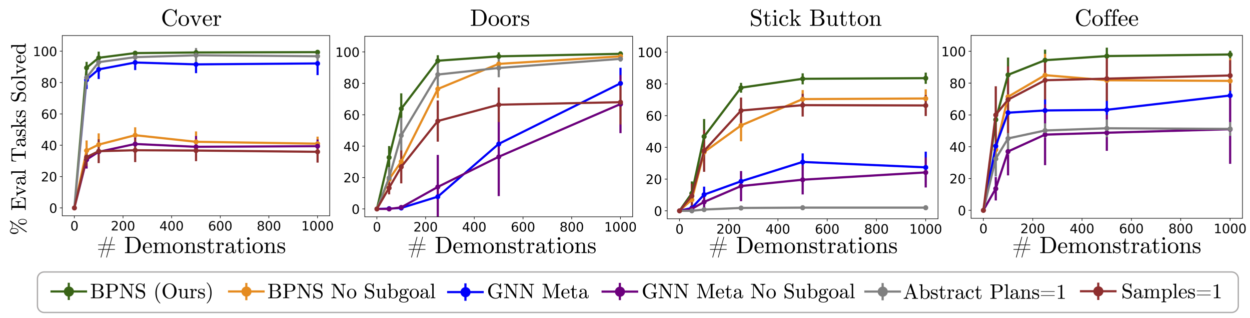


Figure 6.1: **Main results**. Evaluation task success rates versus number of demonstrations. Lines are means and error bars are standard deviation over 10 random seeds. GNN BC is excluded because its performance is consistently very poor, and its training is the most expensive; see the table below.

- *GNN Meta*: Instead of AI planning, this approach learns a graph neural network (GNN) metacontroller [11, 15, 36] 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 [11, 12, 54, 55].
- *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.

6.3 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.

6.4 Results and Analysis

Figure 6.1 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 4.1 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 demon-

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)

Table 6.1: A comparison of approach BPNS with approach GNN BC.

strations 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.

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 [1, 2] (§C.3), learning from human demonstrations (§C.5), the impact of irrelevant predicates (§C.6) or objects (§C.7), the filtering of low-data skills (§C.8), comparison to oracle skills (§C.9), the structure of the learned operators (§C.10), and skill learning with invented predicates (§C.11).

Chapter 7

Related Work

Our work contributes to a long line of research in skill learning for robotics, with prior works considering both reinforcement learning [56–65] and learning from demonstrations [10, 36, 43, 66–71]. In the context of hierarchical RL [30, 31, 72], it is possible to interpret our skill samplers as managers and policies as workers. Our work is also related to *parameterized* skill learning [50–53, 73, 74] if we view subgoals as parameters. In the AI planning literature, prior works use a set of given planning operators to learn skills [9, 11, 12, 54, 55, 75–77]. Achterhold et al. [78] 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 [64], and then learn a metacontoller over the skills. The "GNN Meta" baseline is inspired by this work. Recent work by Cheng and Xu [79] is also similar, in that they employ reinforcement learning instead of behavior cloning to learn the low-level policies for operators in a TAMP framework. Instead of operators, recent works have also used pretrained language models to generate abstract "instructions" and then learn language-conditioned skills [71, 80–82].

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 [1, 47, 83–

87], learning samplers [2, 42, 88–91], or learning predicates [6, 17–21, 92–96]. Our operator and sampler learning algorithms are most directly related to that of Chitnis et al. [2], 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 [46]. Wang et al. [90] 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 [32–34].

Chapter 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.11), 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? (4) Can this approach be extended to more levels of hierarchy? (5) To what extent are KD1 and KD2 applicable in hierarchical approaches that build on large language models? [71, 80–82]. Answering these questions will help us leverage hierarchical planning at scale.

Appendix A

Environment Details

Here we provide detailed descriptions of each of experiment environments. See (§6) for high-level descriptions.

A.1 Cover Environment Details

- **Types:**
  - The `block` type has features `height`, `width`, `x`, `y`, `grasp`.
  - The `target` type has features `width`, `x`.
  - The `gripper` type has features `x`, `y`, `grip`, `holding`.
  - The `allowed-region` type, which is used to determine whether picking or placing at certain positions is allowed, has features `lower-bound-x`, `upper-bound-x`.
- **Action space:** $\mathbb{R}^3$. An action (`dx`, `dy`, `dgrip`) is a delta on the gripper.
- **Predicates:** `Covers`, `HandEmpty`, `Holding`, `IsBlock`, `IsTarget`.
- **Contact-related predicates:** `Covers`, `HandEmpty`, `Holding`.
- **Notes:** This extends the environment of [1, 2, 21] to make the robot move in two dimensions. In the previous work, the environment was referred to as PickPlace1D.

A.2 Doors Environment Details

- **Types:**
    - The `robot` type has features `x`, `y`.
    - The `door` type has features `x`, `y`, `theta`, `mass`, `friction`, `rotation`, `target`, `is-open`.
    - The `room` type has features `x`, `y`.
    - The `obstacle` type has features `x`, `y`, `width`, `height`, `theta`.
- **Action space:** $\mathbb{R}^3$. An action (`dx`, `dy`, `drotation`) is a delta on the robot. The rotation acts on the door handle when the robot is close enough to the door.
- **Predicates:** `InRoom`, `InDoorway`, `InMainRoom`, `TouchingDoor`, `DoorIsOpen`, `DoorInRoom`, `DoorsShareRoom`.
- **Contact-related predicates:** `TouchingDoor`, `InRoom`.
- **Notes:** The rotation required to open the door is a complicated function of the door features.

A.3 Stick Button Environment Details

- **Types:**
    - The `gripper` type has features `x`, `y`.
    - The `button` type has features `x`, `y`, `pressed`.
    - The `stick` type has features `x`, `y`, `held`.
    - The `holder` type has features `x`, `y`.
- **Action space:** $\mathbb{R}^3$. An action (`dx`, `dy`, `z-force`) is a delta on the gripper and a force in the z direction (see notes below).

- **Predicates:** `Pressed`, `RobotAboveButton`, `StickAboveButton`, `AboveNoButton`, `Grasped`, `HandEmpty`.
- **Contact-related predicates:** `Grasped`, `Pressed`.
- **Notes:** Picking and pressing succeed when (1) the `z-force` action exceeds a threshold; (2) there are no collisions; and (3) when the respective objects are close enough in `x`, `y` space.

A.4 Coffee Environment Details

- **Types:**
  - The `gripper` type has features `x`, `y`, `z`, `tilt-angle`, `wrist-angle`, `fingers`.
  - The `pot` type has features `x`, `y`, `rotation`, `is-held`, `is-hot`.
  - The `plate` type has feature `is-on`.
  - The `cup` type has features `x`, `y`, `liquid-capacity`, `liquid-target`, `current-liquid`.
- **Action space:** $\mathbb{R}^6$. An action (`dx`, `dy`, `dz`, `dtilt`, `dwrist`, `dfingers`) is a delta on the gripper.
- **Predicates:** `CupFilled`, `PotOnPlate`, `Holding`, `ButtonPressed`, `OnTable`, `HandEmpty`, `PotHot`, `RobotAboveCup`, `PotAboveCup`, `NotAboveCup`, `PressingButton`, `Twisting`.
- **Contact-related predicates:** `Holding`, `HandEmpty`, `CupFilled`, `ButtonPressed`.
- **Notes:** The liquid in a cup increases when a full pot is tilted and close enough to the cup.

Appendix B

Approach Details

Here we provide detailed descriptions of each approach evaluated in experiments. See (§6) for high-level descriptions.

B.1 Bilevel Planning with Neuro-Symbolic Skills (BPNS)

BPNS is our main approach, as described in the main paper.

Planning: The number of abstract plans $N_{\text{abstract}} = 8$ for Cover and Doors, and $N_{\text{abstract}} = 1000$ for Coffee and Stick Button. We would not expect performance to substantially improve for Cover or Doors with a larger N_{abstract}, since we know that the first abstract plan is generally refinable in these environments; the smaller number was selected for the sake of experiments finishing faster. The number of samples per step $N_{\text{samples}} = 10$ for all environments.

Operator Learning: Operators whose skill datasets comprise less than 1% of the overall number of segments are filtered out. This filtering is helpful to speed up learning and planning in cases where there are rare effects or simulation noise in the demonstrations.

Policy Learning: Policies are fully-connected neural networks with two hidden layers of size 32. Models are trained with Adam for 10,000 epochs with a learning rate of 1e$-$3 with MSE loss.

Sampler Learning: Following Chitnis et al. [2], each sampler consists of two neural networks: a generator and a discriminator. The generator outputs the mean and diagonal covariance of a Gaussian, using an exponential linear unit (ELU) to assure PSD covariance. The generator is a fully-connected neural network with two hidden layers of size 32, trained with Adam for 50,000 epochs with a learning rate of 1e$-$3 using Gaussian negative log likelihood loss. The discriminator is a binary classifier of samples output by the generator. Negative examples for the discriminator are collected from other skill datasets. The classifier is a fully-connected neural network with two hidden layers of size 32, trained with Adam for 10,000 epochs with a learning rate of 1e$-$3 using binary cross entropy loss. During planning, the generator is rejection sampled using the discriminator for up to 100 tries, after which the last sample is returned.

B.2 BPNS No Subgoal

BPNS No Subgoal is a variation of BPNS that does not use subgoal parameterization.

Planning: N_{abstract} is the same as BPNS and N_{samples} is not applicable.

Learning: Operator learning is the same as BPNS. For policy learning, for each input $x \circ y$ in the training data, where y is the subgoal, we use x instead, i.e., the state alone. No samplers are learned.

B.3 Graph Neural Network Metacontroller (GNN Meta)

GNN Meta is a mapping from state, abstract state, and goal to a ground skill. This baseline offers a learning-based alternative to AI planning in the outer loop of bilevel planning.

Planning: Repeat until the goal is reached: query the model on the current state, abstract state, and goal to get a ground skill. Invoke the ground skill's sampler up to 100 times to find a subgoal that leads to the abstract successor state predicted by the skill's operator. If successful, simulate the state forward; otherwise, terminate with failure.

Learning: Skill learning is identical to BPNS. This approach additionally learns a meta-controller in the form of a GNN. Following the baselines presented in prior work [2], the GNN is a standard encode-process-decode architecture with 3 message passing steps. Node and edge modules are fully-connected neural networks with two hidden layers of size 16. We follow the method of Chitnis et al. [2] for encoding object-centric states, abstract states, and goals into graph inputs. To get graph outputs, we use node features to identify the object arguments for the skill and a global node with a one-hot vector to identify the skill identity. The models are trained with Adam for 1000 epochs with a learning rate of 1e$-$3 and batch size 128 using MSE loss.

B.4 GNN Meta No Subgoal

GNN Meta No Subgoal is the same as GNN Meta, but with skills learned via BPNS No Subgoal.

B.5 GNN Behavioral Cloning (GNN BC)

GNN BC is a mapping from states, abstract states, and goals, directly to actions. This approach is model-free; at evaluation time, it is queried at each state, and the returned action is executed in the environment. The GNN architecture and training is identical to GNN Meta, except that output graphs consist only of a global node, which holds the fixed-dimensional action vector.

B.6 Samples=1

This ablation is identical to BPNS, except with $N_{\text{samples}} = 1$ during planning.

B.7 Abstract Plans=1

This ablation is identical to BPNS, except with $N_{\text{abstract}} = 1$ during planning.

Appendix C

Additional Results

Here we present additional results to supplement the main results in (§6).

C.1 Planning Time Analysis

Figure C.1 reports evaluation task success rate as a function of planning time for BPNS, Samples=1, and Abstract Plans=1. In Cover and Doors, performance peaks within the first few seconds of wall-clock time. This is consistent with our finding that the first abstract plan is generally refinable in these two environments. In Stick Button and Coffee, performance increases more gradually for BPNS over time. Furthermore, given a small time budget, the Samples=1 ablation sometimes solves more evaluation tasks than BPNS. In these two environments, the first abstract plan is typically not refinable: BPNS exhaustively attempts to sample that abstract plan and others before arriving at a refinable abstract plan. With Samples=1, the unrefinable abstract plans are quickly discarded after one sampling attempt. The gap that later emerges between BPNS and Samples=1 is due to tasks where one sampling attempt of a refinable abstract plan is not enough. This trend is fairly specific to the details of our bilevel planning implementation. Other search-then-sample TAMP techniques that

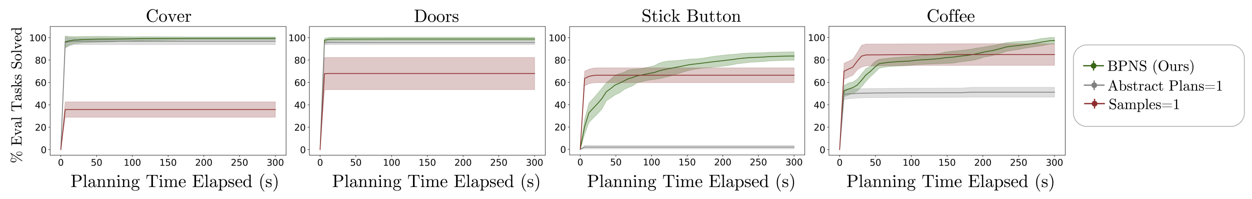


Figure C.1: **Planning time analysis**. Evaluation task success rate as a function of planning time elapsed for BPNS and the two ablations. All results are over 10 random seeds and all models are trained on 1000 demonstrations. Lines are means and shaded areas are one standard deviation.

do not exhaustively sample abstract plans before moving onto the next one may converge faster [3].

C.2 GNN Meta Additional Analysis

We were initially surprised by the poor performance of GNN Meta in Stick Button and Coffee, given that the target functions are intuitively straightforward. In Stick Button, the model should be able to use the positions of the buttons in the low-level state to determine whether they can be directly reached, or if the stick should be used instead. In Coffee, the model should be able to use the rotation of the pot to determine if it must be first rotated. To explain the poor performance of GNN Meta in these environments, we hypothesized that the model struggles to extrapolate to more objects than seen during training. We tested this hypothesis by evaluating the same GNN Meta models on new tasks from Stick Button and Coffee, but with object counts from the training distribution. Figure 6.1 shows that the performance of GNN Meta sharply improves, supporting the hypothesis.

C.3 Comparison to [1, 2]

Here we elaborate on the relationship between this work and our prior work [1, 2]. In brief, [2] extends [1] by removing the assumption that samplers are given. Our work here extends

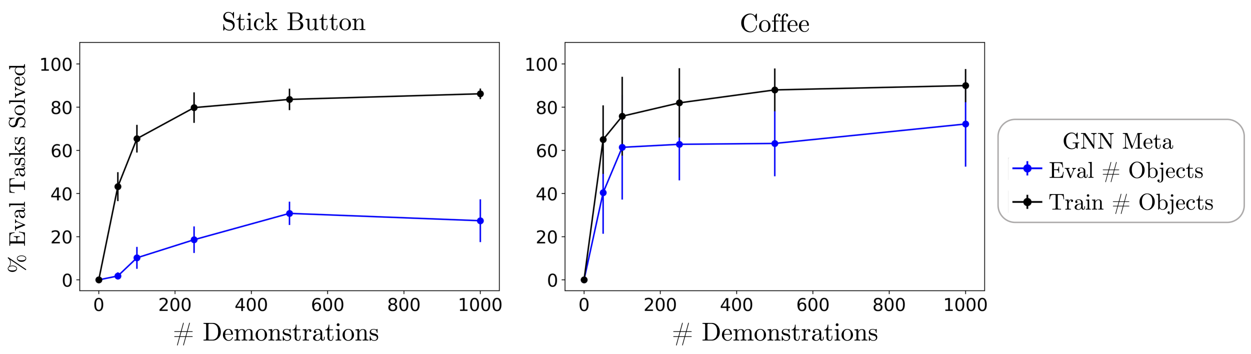


Figure C.2: **GNN Meta additional results**. Task success rates for the GNN Meta baseline on tasks with more objects (Eval) or the same number of objects (Train) as seen during training. The gap suggests that the poor performance of GNN Meta in the main results (Figure 6.1) is largely attributable to a failure to generalize over object count. All results are over 10 random seeds. Lines are means and error bars are standard deviations. Note that in Cover and Doors, the distribution of object number is the same between train and evaluation.

[2] by removing the assumptions that high-level controllers are given (e.g., pick, place, move), and that demonstration data is provided in terms of those high-level controllers.

Removing the assumption that high-level controllers are given requires several nontrivial steps. First, in [1, 2], the demonstration data is given in terms of high-level controllers: each transition corresponds to the execution of an entire controller, and the controller identity is known. This setting makes operator learning much easier because each transition corresponds to exactly one operator. The controller identity is also used to make operator learning easier. In contrast, we are given demonstration data where the actions are low-level (e.g., end effector positions of the robot), and we seek to learn operators that correspond to the execution of *many* low-level actions in sequence. This necessitates segmentation. Second, because the controllers are fully defined in the previous work, including their continuous parameterizations, it is straightforward to set up the sampler learning problems. In contrast, we have no such continuous controller parameterization given to us in this work. One of our main insights is that subgoal states can be used as the basis for continuous parameterization. This insight follows from KD1 and has the benefit that we can automatically derive targets for learning from the demonstration data. Finally, we must learn the controllers themselves.

Approach	**Tasks Solved**
Ours	99.40 (0.64)
Ablation [12]	2.80 (1.20)

Table C.1: Comparing approach BPNS to an ablation that is the method of [2].

In the previous work, these controllers were hardcoded.

With these differences in mind, to motivate our work, we designed a version of our approach that ablates policy learning, and can be seen as an application of [2]. This ablation works as follows:

- For each transition in the demonstration data, we create one (single-step) segment.
- Partitioning and lifting are unmodified. Operator learning is also unmodified.
- Instead of policy learning, we create a "pass-through" policy architecture that consumes a continuous action (instead of a subgoal) and returns that action.
- Sampler learning is unmodified, except rather than sampling subgoals, we sample actions to give to the pass-through policy, consistent with the sampler learning of [2].

Another way to understand this ablation is that we are applying the method of [2] but supposing that the given action space consists of a single parameterized controller, with the parameter space equal to the low-level action space. We ran this ablation in the Cover environment with 1000 demonstrations and hyperparameters unchanged from our main experiments. Results are shown in Table C.1, with the numerical entries representing means (standard deviations) over 10 seeds.

Qualitatively, we see that the learned skills are the same between the two approaches, except that the ablation learns an additional skill with empty operator preconditions and effects. This additional skill is important and provides insight into the discrepancy between the two approaches. In the demonstration data, the majority of transitions do not correspond to any change in the abstract state. For example, as the robot moves in preparation for a

Environment	**Train Version**	**Test Version**	**Success Rate**
Stick Button	Easy	Hard	83.60 (1.78)
Stick Button	Hard	Easy	98.20 (0.94)
Coffee	Easy	Hard	98.00 (1.18)
Coffee	Hard	Easy	98.20 (1.04)

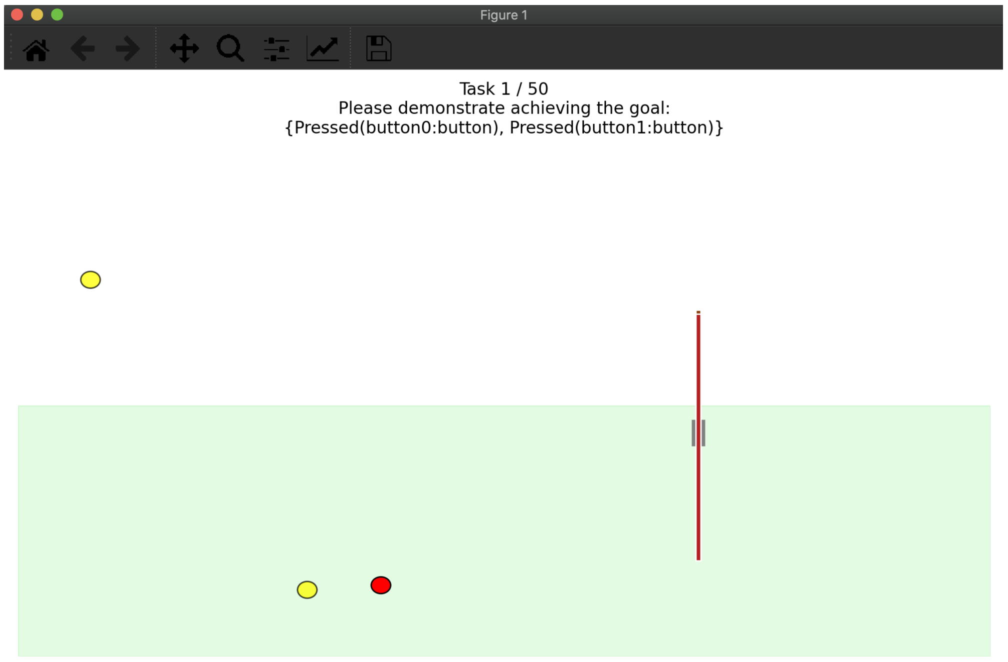
Table C.2: Training on hard and evaluating on easy tasks.

pick or place, the abstract state is constant. These transitions lead to the empty operator effects. The preconditions are empty because there is nothing in common between the cases where no abstract effect occurs — sometimes the robot is holding an object, and sometimes it is not. This empty skill makes abstract planning very difficult because there is no signal for the planner to realize when using the skill would bring the robot closer to the goal. Note, though, that this empty skill is needed for planning, and simply removing it would make performance even worse; the remaining two skills can only handle the single step immediately preceding the respective effects (i.e., the step where an object is picked or placed).

C.4 Train on Hard, Evaluate on Easy

In this additional experiment, rather than training on easy tasks and evaluating on hard tasks, we instead train on hard tasks and evaluate on easy ones. Specifically, we consider swapping the number of objects in the Stick Button and Coffee environments, where there is a difference in the object distribution between train and evaluation. We ran this experiment with 1000 demonstrations and hyperparameters unchanged from our main experiments. Results are shown in Table C.2. We find that performance with swapped object distributions matches or exceeds that with the original distributions. This is consistent with the intuition that generalizing from smaller to larger problems is usually more difficult than the opposite.

C.5 Learning from Human Demonstrations



Task 1 / 50
Please demonstrate achieving the goal:
{Pressed(button0:button), Pressed(button1:button)}

Figure C.3: GUI for collecting human demonstrations in Stick Button.

We ran an additional experiment with human demonstrations. To collect the demonstrations, we created a graphical user interface (GUI) and a simplified version of the Stick Button environments. The GUI is shown in Figure C.3. Here the robot is a red circle, the buttons are yellow circles, the stick is a brown rectangle, the stick holder is the gray rectangle, and all buttons outside the green region are unreachable by the robot. Clicking a point on the screen initiates a translational robot movement, with the magnitude clipped so that the robot can only move so far in one action. Pressing any key initiates a grasp of the stick if the robot is close enough, or a press of the button if either the robot or stick head is close enough.

We used the GUI to collect 994 human demonstrations. We then ran BPNS with hyperparameters identical to the main results. Results are shown in Table C.3. We find that the number of evaluation tasks solved by the approach trained on human demonstrations is slightly below that of automated demonstrations. This small gap can be attributed to

Demos	% Tasks Solved	Test Time (s)	Nodes Created
Automated	83.60	51.80	421.51
Human	80.00	40.07	430.93

Table C.3: Human demonstration results in Stick Button.

noise, suboptimality, and inconsistencies in the human demonstrations. Overall, the strong performance of BPNS on human demonstrations suggests that the approach can be scaled.

C.6 Impact of Irrelevant Predicates

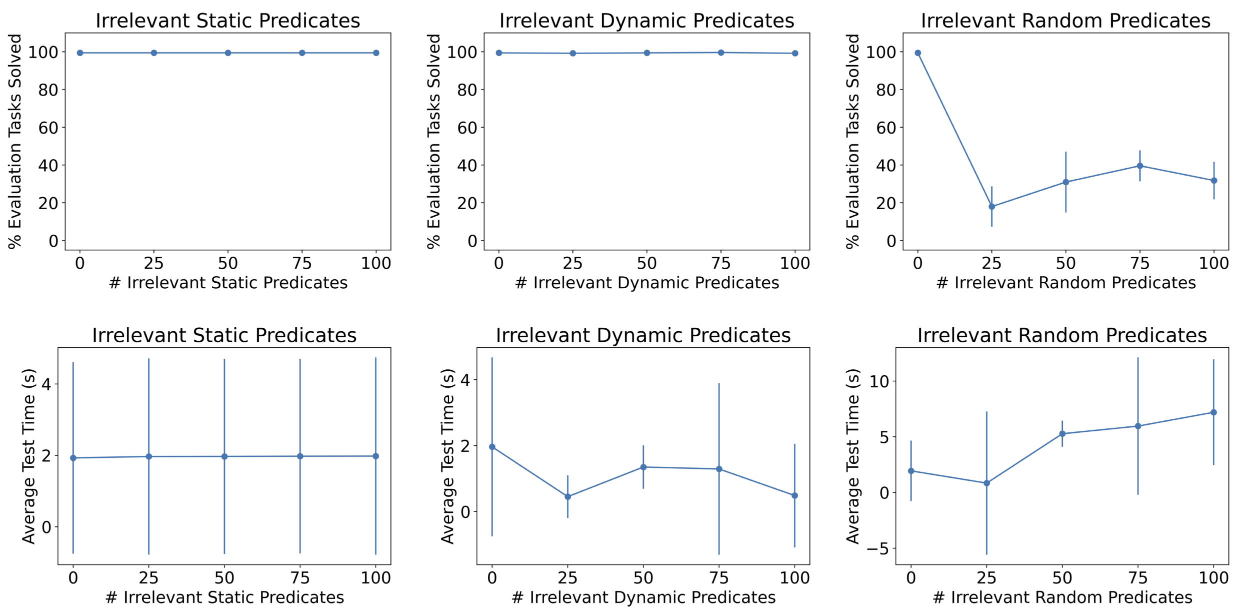


Figure C.4: Impact of irrelevant predicates.

We conducted three additional experiments in the Cover environment to investigate the influence of irrelevant predicates. We used 1000 demonstrations and all hyperparameters are the same as in the main results.

First, we added a variable number of *static* predicates, i.e., predicates whose evaluation is always True or False for an object regardless of the low-level state. Second, we added a variable number of *dynamic* (i.e., not static) predicates. Concretely, we created predicates that randomly threshold the y position of the robot. Third, in an attempt to create a setting that is adversarially bad for our framework, we added a variable number of *random* predicates, where the evaluation of each predicate is completely random on each input, with 50% probability of being True and without regard for the low-level state.

Results for each of the three experiments are shown in Figure C.4. Static predicates have no apparent impact on evaluation performance. Qualitatively, we see that the preconditions of the learned operators have more preconditions, corresponding to the static predicates that are always True. The form of the operators, and the rest of the learned skills, are otherwise unchanged. The dynamic predicates also have little to no impact on evaluation performance. The learned operators again have additional preconditions, but also have additional dynamic predicates in the effects. However, these dynamic predicates are relatively "well-behaved", whereas in more complicated environments, predicate evaluations could be much less regular. This motivates the random predicates experiments, where indeed we see a substantial drop in evaluation performance. This drop is precipitated by a much larger and more complex set of learned operators, which makes abstract planning and learning more difficult. Altogether, these results confirm that the choice of predicates is important.

C.7 Impact of Irrelevant Objects

We conducted two additional experiments in the Cover environment to test the influence of irrelevant objects. We used 1000 demonstrations and all hyperparameters the same as in the main results.

In the first experiment, we added a variable number of irrelevant blocks during training, and in the second experiment, we added them instead during evaluation. The blocks are irrelevant because they are not involved in goals; they are also placed off the table so as

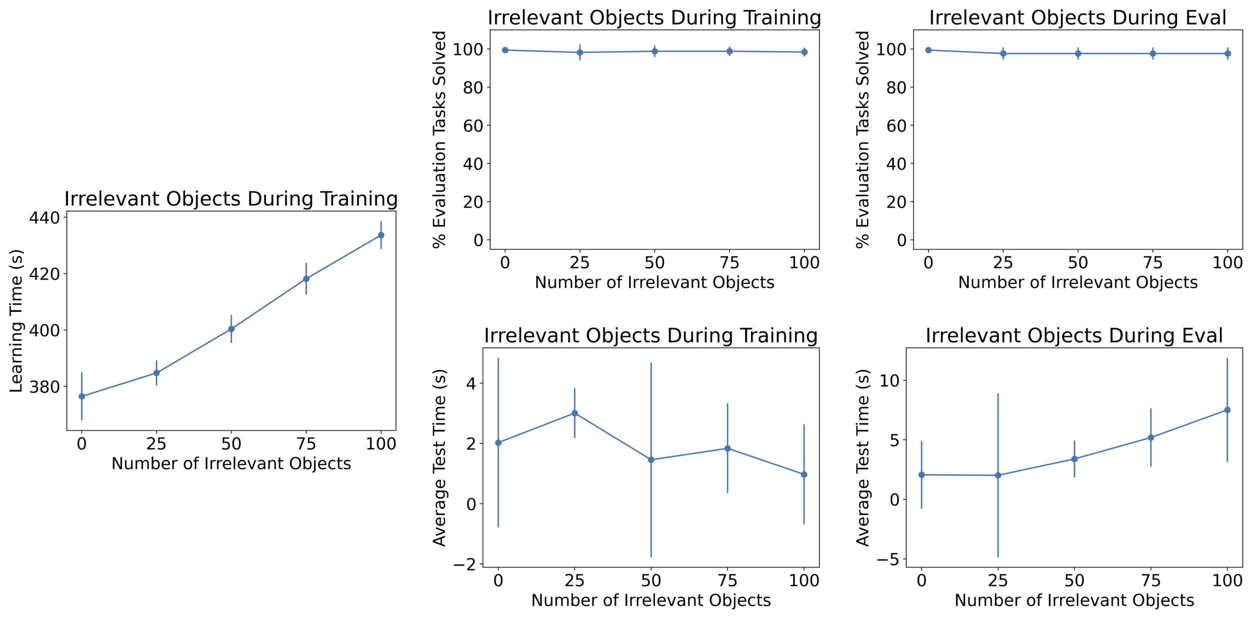


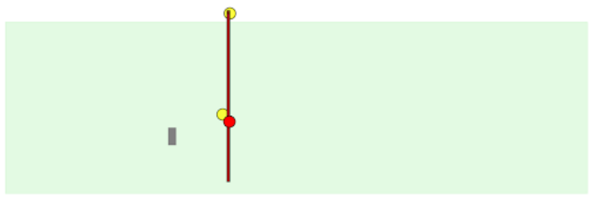
Figure C.5: Impact of irrelevant objects.

to not cause collisions with the original blocks. The blocks *do* have the same type as the original blocks, so they will not be simply filtered out during type matching.

Results for each of the experiments are shown in Figure C.5. Adding the irrelevant objects during training has no impact on evaluation performance. This is expected, since our preprocessing pipeline naturally filters out the irrelevant objects. The irrelevant objects exhibit a small impact on learning time due to the cost of evaluating predicates. Adding the irrelevant objects during evaluation has a small impact on evaluation performance, although success rate is robust with up to 100 irrelevant objects. The increase in evaluation time is due to predicate evaluation and an increased branching factor during abstract planning.

C.8 Disabling Filtering of Low-Data Skills

We ran an additional experiment where we disabled the filtering out of skills with low data. We used 1000 demonstrations



Environment	Filter?	% Tasks Solved	Test Time (s)	Nodes Created
Cover	Yes	99.40 (0.64)	2.02 (2.84)	7.15 (0.42)
Cover	No	99.40 (0.64)	1.99 (2.79)	7.30 (0.59)
Doors	Yes	98.80 (0.80)	1.33 (0.59)	41.94 (4.56)
Doors	No	98.80 (0.80)	1.70 (1.05)	50.84 (24.27)
Stick Button	Yes	83.60 (1.78)	51.80 (11.47)	421.51 (81.28)
Stick Button	No	23.80 (9.68)	121.43 (57.14)	2725.17 (1898.47)
Coffee	Yes	98.00 (1.18)	49.39 (10.22)	53.68 (7.07)
Coffee	No	98.20 (1.04)	47.72 (9.52)	53.95 (7.01)

Table C.4: Disabling filtering of low-data skills.

and identical hyperparameters to our main experiments. Results are shown in Table C.4, with the numerical entries representing means (standard deviations) over 10 seeds. The results show that evaluation performance in Cover, Doors, and Coffee is largely unaffected by filtering, while the performance in Stick Button is substantially affected. The performance in Stick Button can be traced back to the rare situation illustrated in the image on the right. Typically, when the robot (red) presses a button (yellow) with the stick (brown), the robot is not above any other button. However, in this case, the robot is coincidentally above a second button while executing the stick press. This leads to an operator with an effect set that includes both the button being pressed and the robot being above a second button. That operator is ultimately detrimental to planning because in the vast majority of cases, it is not possible for the robot to press the button while being above a second button, so refining this operator usually fails. This operator also has a very small amount of training data, which makes the associated policy and sampler unreliable. For similar reasons, we prefer to filter out skills with too little training data by default.

Env	Approach	% Tasks Solved	Test Time (s)	Nodes Created
Cover	Ours	99.40 (0.64)	2.02 (2.84)	7.15 (0.42)
Cover	Oracle	98.80 (0.30)	0.03 (0.01)	7.01 (0.17)
Doors	Ours	98.80 (0.80)	1.33 (0.59)	41.94 (4.56)
Doors	Oracle	100.00 (0.00)	1.04 (0.09)	84.12 (20.17)
Stick Button	Ours	83.60 (1.78)	51.80 (11.47)	421.51 (81.28)
Stick Button	Oracle	90.40 (1.99)	0.18 (0.03)	320.32 (46.92)
Coffee	Ours	98.00 (1.18)	49.39 (10.22)	53.68 (7.07)
Coffee	Oracle	100.00 (0.00)	0.07 (0.01)	67.25 (8.12)

Table C.5: Comparison to oracle skills.

C.9 Comparison to Oracle

We collected statistics for an oracle approach that uses manually-designed skills. We use identical hyperparameters to the main results. The statistics are reported in Table C.5, with the numerical entries representing means (standard deviations) over 10 seeds, and where "Ours" is the main approach, BPNS, trained with 1000 demonstrations. In Doors and Coffee, the learned skills require fewer node creations during abstract search to find a plan. This difference can be attributed to (1) overly general preconditions in the operators of our manually designed skills and (2) more targeted sampling when using the learned samplers versus manually designed samplers. In all environments, the wall-clock time taken to plan with our learned skills is far greater than that of the oracle. From profiling, we can see that this difference is largely due to neural network inference time in both the learned samplers and learned skill policies. In contrast, the manually designed skills are written in pure Python, and can therefore be evaluated very efficiently.

C.10 Learned Operator Examples

See Figure C.6 for examples of learned operators in each environment. Below, we describe the operators that are typically learned at convergence for each of the environments. These descriptions are based on inspection of the operator syntax, speaking to their interpretability.

- **Cover:**
  1. Pick up a block.
  2. Place a block on a target.
- **Doors:**
  1. Move to a door from the main part of a room (not in a doorway).
  2. Move to a door from another doorway.
  3. Move through an open door.
  4. Open a door.
- **Stick Button:**
  1. Move from free space to press a button with the gripper.
  2. Move from above another button to press a button with the gripper.
  3. Move from free space to pick up a stick.
  4. Move from above a button to pick up a stick.
  5. Move from free space to press a button with the stick.
  6. Move from above another button to press a button with the stick.
- **Coffee:**
  1. Pick up the coffee pot.
  2. Put the coffee pot on the hot plate.

Metric	**Manual**	**Learned**
% Eval Tasks Solved	99.40 (0.64)	99.20 (0.92)
# Predicates	5.00 (0.00)	5.40 (0.80)
Eval Time (s)	2.00 (2.78)	2.82 (3.10)
Learning Time (s)	372.01 (4.55)	4898.16 (692.09)

Table C.6: Results in the Cover environment when learning predicates.

3. Turn on the hot plate.
4. Move from above no cup to pour into a cup.
5. Move from above another cup to pour into a cup.
6. Twist the coffee pot.
7. Pick up the coffee pot after twisting.

C.11 Predicate Invention Preliminary Results

We ultimately envision a continually learning robot that uses symbols to learn skills and skills to learn symbols in a cycle of self-improvement. One plausible path toward realizing this vision would start with a set of manually designed symbols, as we did in this work, or skills, as done by Konidaris et al. [6]. Alternatively, we could start with demonstrations alone. In this case, we need to answer the chicken-or-the-egg question: which should be learned first, skills or symbols? Here we present very preliminary results suggesting the viability of learning symbols (predicates) first, and then skills from those learned symbols.

We follow the approach of Silver et al. [21], which starts with a minimal set of *goal predicates* that are sufficient for describing the task goals, and then uses demonstrations to invent new predicates. Specifically, we focus on the Cover environment, where there is only one goal predicate: `Covers`. Given 1000 demonstrations, after learning predicates (see [21] for a description of the approach and note that clustering segments to induce operators was done based on the unique effects observed in segments' effects), we run BPNS skill learning and planning, with the configuration identical to the main experiments. Results are shown in Table C.6. Each entry is a mean (standard deviation) over 10 random seeds. The Manual column uses the manually designed predicates from our main experiments and the Learned column uses learned predicates. Further investigation is needed, but the results do suggest that learning skills on top of learned predicates is a viable direction. We also report the number of predicates learned, evaluation time, and learning time. Consistent with the prior work, we see that additional predicates can be learned that sometimes lead to faster planning during evaluation. We also see that the time bottleneck for the overall system is predicate invention, not skill learning.

```
Learned-Op0:
  Parameters: [?x0:block, ?x1:gripper]
  Preconditions: [HandEmpty(), IsBlock(?x0:block)]
  Add Effects: [Holding(?x0:block, ?x1:gripper)]
  Delete Effects: [HandEmpty()]
```

Example learned operator for picking a block in *Cover*.

```
Learned-Op0:
  Parameters: [?x0:door, ?x1:robot]
  Preconditions: [InDoorway(?x1:robot, ?x0:door),
                  TouchingDoor(?x1:robot, ?x0:door)]
  Add Effects: [DoorIsOpen(?x0:door)]
  Delete Effects: [TouchingDoor(?x1:robot, ?x0:door)]
```

Example learned operator for opening a door in *Doors*.

```
Learned-Op0:
  Parameters: [?x0:gripper, ?x1:stick]
  Preconditions: [AboveNoButton(),
                 HandEmpty(?x0:gripper)]
  Add Effects: [Grasped(?x0:gripper, ?x1:stick)]
  Delete Effects: [HandEmpty(?x0:gripper)]
```

Example learned operator for grasping the stick in *Stick Button*.

```
Learned-Op0:
  Parameters: [?x0:cup, ?x1:pot, ?x2:gripper]
  Preconditions: [Holding(?x2:gripper, ?x1:pot),
                  PotHot(?x1:pot),
                  NotAboveCup(?x2:gripper, ?x1:pot)]
  Add Effects: [CupFilled(?x0:cup),
                PotAboveCup(?x1:pot, ?x0:cup),
                RobotAboveCup(?x2:gripper, ?x0:cup)]
  Delete Effects: [NotAboveCup(?x2:gripper, ?x1:pot)]
```

Example learned operator for pouring in *Coffee*.

Figure C.6: **Learned operator examples**.

Bibliography

[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] Y. Wang, N. Figueroa, S. Li, A. Shah, and J. Shah. Temporal logic imitation: Learning plan-satisficing motion policies from demonstrations, 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] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5 (2):115–135, 1974. ISSN 0004-3702. doi:https://doi.org/10.1016/0004-3702(74)90026-5. URL https://www.sciencedirect.com/science/article/pii/0004370274900265.

[28] F. Bacchus and Q. Yang. The downward refinement property. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, page 286–292, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. ISBN 1558601600.

[29] 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.

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

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

[32] 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.

[33] 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.

[34] 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.

[35] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Learning neuro-symbolic skills for bilevel planning. In K. Liu, D. Kulic, and J. Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 701–714. PMLR, 14–18 Dec 2023. URL https://proceedings.mlr.press/v205/silver23a.html.

[36] 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.

[37] 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.

[38] 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.

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

[40] 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.

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

[42] 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.

[43] 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.

[44] 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.

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

[46] 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.

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

[48] 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.

[49] 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.

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

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

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

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

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

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

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

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

[62] 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.

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

[64] 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.

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

[66] 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.

[67] 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.

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

[69] 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.

[70] 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.

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

[72] 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.

[73] 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.

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

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

[76] 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.

[77] 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.

[78] 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.

[79] S. Cheng and D. Xu. League: Guided skill learning and abstraction for long-horizon manipulation, 2023.

[80] 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.

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

[82] 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.

[83] 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.

[84] 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.

[85] 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.

[86] 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.

[87] 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.

[88] 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.

[89] 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.

[90] 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.

[91] 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.

[92] 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.

[93] 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.

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

[95] 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.

[96] 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.