Practice Makes Perfect: Planning to Learn Skill Parameter Policies

Nishanth Kumar*†‡, Tom Silver*†‡, Willie McClinton†‡, Linfeng Zhao†§,
Stephen Proulx†, Tomás Lozano-Pérez‡, Leslie Pack Kaelbling‡ and Jennifer Barry†
†The AI Institute, ‡MIT CSAIL, §Northeastern University

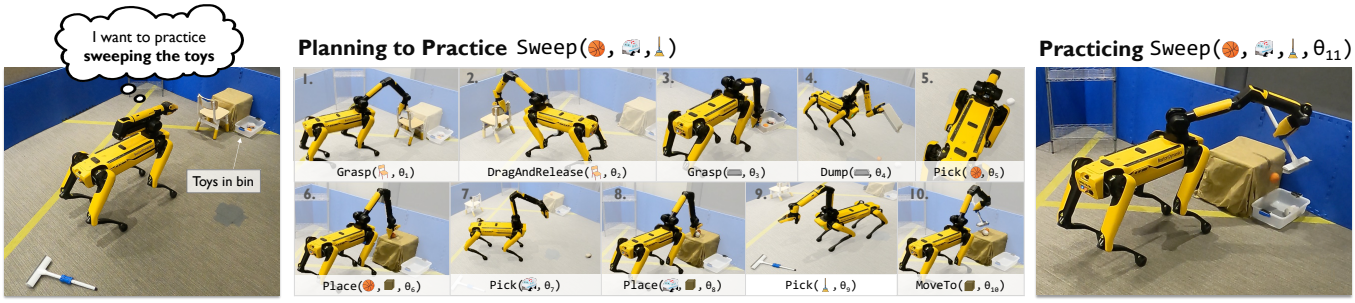

Fig. 1: **Planning to practice.** To practice a skill, the robot needs to be in a state where the skill can be initiated. Here, the robot plans to practice sweeping two toys into a bin from its initial state (left) in the Cleanup Playroom environment. This requires chaining up to 19 skills (middle), some of which are omitted for brevity, before practicing (right). Some skill object parameters are also omitted.

***Abstract*—One promising approach towards effective robot decision making in complex, long-horizon tasks is to sequence together *parameterized skills*. We consider a setting where a robot is initially equipped with (1) a library of parameterized skills, (2) an AI planner for sequencing together the skills given a goal, and (3) a very general prior distribution for selecting skill parameters. Once deployed, the robot should rapidly and autonomously learn to improve its performance by specializing its skill parameter selection policy to the particular objects, goals, and constraints in its environment. In this work, we focus on the active learning problem of choosing which skills to *practice* to maximize expected future task success. We propose that the robot should *estimate* the competence of each skill, *extrapolate* the competence (asking: "how much would the competence improve through practice?"), and *situate* the skill in the task distribution through competence-aware planning. This approach is implemented within a fully autonomous system where the robot repeatedly plans, practices, and learns without any environment resets. Through experiments in simulation, we find that our approach learns effective parameter policies more sample-efficiently than several baselines. Experiments in the real-world demonstrate our approach's ability to handle noise from perception and control and improve the robot's ability to solve two long-horizon mobile-manipulation tasks after a few hours of autonomous practice. Project website: https://planning-to-practice.github.io**

I. Introduction

Given the recent progress in robot skill learning and design [11, 22, 39, 59, 65], we are quickly approaching a future where robots will arrive at their deployment sites equipped with a library of general-purpose skills. Each robot will sequentially compose these skills in different ways to accomplish long-horizon tasks that will vary considerably between deployment sites. As the robot gathers experience during deployment, it should get better over time. In particular, the robot should learn to *rapidly specialize* its skills to the unique objects, goals, and constraints that it repeatedly encounters during deployment.

In this work, we consider skills that are continuously parameterized and we focus on *parameter policy learning* [1, 15, 20, 40, 44] as a mechanism for rapidly specializing skills. For example, a "pick" skill may be parameterized by a relative grasp and a "sweep" skill by a sweeping velocity (Figure 1). Starting from general-purpose priors [7, 26, 47, 54], we want the robot to quickly learn specialized policies for selecting grasps, push velocities, and other skill parameters. Following previous work [1, 33, 52], we consider parameterized skills that are (extended) *options* [57]; each skill has an initiation condition, a parameterized controller, a termination condition, and a success condition. For example, a "place" skill can be initiated when the robot is holding an object and facing a surface; the skill terminates after the robot opens its gripper; and the skill is successful if the object is subsequently stably resting on the surface. Options are closely related to AI planning operators [33, 52] and we can leverage this relationship to efficiently *plan* a sequence of skills to reach a goal [28].

We consider parameter policy learning in the context of reset-free online learning [25, 38, 42, 58] where the robot alternates between solving given tasks (*task time*) and taking actions of its choosing (*free time*). For example, a given task might be to "clear objects off the table" (Figure 1). We focus on free time and ask: how should the robot select actions so that, after learning parameter policies from the collected experience, the likelihood of solving given tasks in the future is maximized? This is an *embodied active learning* problem [16, 36, 42, 45], which is distinct from standard active learning [49] in that the robot must reason sequentially. For example, to collect one "sweep" data point in our experiments,

*Equal contribution. Work done during internship at The AI Institute. Correspondence to {njk, tslvr}@csail.mit.edu.

the robot needs to execute up to 19 skills in sequence to reach a state where sweeping is possible (Figure 1). This setting is also related to *exploration* in the reinforcement learning literature [2, 8, 13, 29, 46]; we consider baselines from that literature in experiments. Compared to end-to-end RL though, our setting has significantly more structure, which we can leverage to achieve much more sample-efficient learning, especially over long-horizon tasks.

To learn parameter policies through embodied active learning, we consider *planning to practice parameterized skills*. During free time, the robot repeatedly selects a skill, plans to a state where that skill can be initiated, practices the skill (selects continuous parameters to try), records the success condition outcome, and then updates its parameter policy accordingly. The central question is: how should the robot decide what skills to practice? One approach would be to practice the skill with the lowest *competence* [14, 56], that is, the skill that most often fails to achieve its success condition. But that skill may be impossible to improve or irrelevant to the given tasks.

We propose that the robot should instead *practice the skill whose predicted competence improvement would maximally benefit the overall task distribution*. Implementing this skill selection strategy requires three steps: *estimating* current skill competence; *extrapolating* the competence by predicting how much it would hypothetically improve through practice; and *situating* the competence in the task distribution by predicting how overall task success rates would hypothetically improve. We propose a Beta-Bernoulli time series model to estimate and extrapolate skill competence and use cost-aware AI planning [28] to situate the competence in the task distribution.

In experiments, we evaluate the extent to which our **Estimate, Extrapolate & Situate (EES)** approach enables the robot to make efficient use of its free time as measured by its success rate during task time. In three simulated environments, we compare to seven baselines and find that EES is consistently the most sample-efficient. We also implement EES in two real mobile manipulation environments using a Boston Dynamics Spot robot with an arm (Figures 1, 2). In these environments, the robot plans and practices autonomously for several hours, coping with noise inherent to real-world perception and control, and rapidly improves its ability to solve long-horizon mobile-manipulation tasks.

II. Problem Setting

This paper proposes a method for active practicing in the context of a robot system that has mechanisms for planning and learning. In this section, we describe our problem setting, including assumptions about the robot's environment as well as minimal specifications for its planning and learning modules.

A. Modelling the World

We assume that the robot and its environment are modelled as a goal-based Markov Decision Process with object-oriented states [17] and parameterized actions [1, 40]. States are factored into objects and their continuous features. For example, consider the Ball-Ring environment shown in Figure 2. The `ball`, `ring`, `table`, `floor`, and `robot` itself are objects, and their features include, for example, *gripper joint value* (for the robot) and *xyz position* (for other objects). Of course, a real-world robot cannot perceive such features directly, so we assume that the robot is equipped with a perception system that can construct a fully-observed state $x_t \in \mathcal{X}$ from raw sensory observations at each time step $t \in \mathbb{Z}^+$. This model does not account for the perception noise that exists in the real world, but in experiments, we find that our approach is reasonably robust to that noise (Section IV).

The action space of the MDP is defined by a set of *parameterized skills* that have continuous parameters. It is often convenient to define *object parameters* as well, but for the purpose of this work, these play a minor role, so we treat them as part of the skill except where otherwise noted. For example, in the Ball-Ring environment, `Place(ball, table, ∘)` is one skill $u \in \mathcal{U}$, where $\circ$ denotes a placeholder for a continuous parameter, and `Place(ring, floor, ∘)` is another $u' \in \mathcal{U}$. The continuous parameters for both skills are *xy* relative offsets between the gripper and target surface (the height and gripper orientation are fixed).

To define skills formally, we use an extension of the options framework [57]. A parameterized skill $u \in \mathcal{U}$ is given by a tuple $(I, \Theta, \mu, \beta, J)$ where $I : \mathcal{X} \to \{0, 1\}$ characterizes states where the skill can be initiated, $\Theta \subseteq \mathbb{R}^m$ is the set of possible continuous parameters, $\mu(x, \theta)$ is a low-level controller that takes a state x and continuous parameters $\theta \in \Theta$ as input, $\beta : \mathcal{X} \to \{0, 1\}$ is a termination condition, and $J : \mathcal{X} \to \{0, 1\}$ is a success condition indicating whether the skill has achieved its intended outcome in the terminal state. A skill with parameters assigned is treated as an atomic action. After an action $a_t \in \mathcal{A}$ is executed, the environment advances according to an unknown transition distribution $x_{t+1} \sim P(\cdot \mid x_t, a_t)$.

The robot is tasked with achieving particular *goals*. Each goal is sampled from a task distribution $g \sim P(\cdot \mid x_0)$. For example, in the Ball-Ring environment, the goal might be that the ball is stably at rest on the table. We do not assume direct access to the goal distribution; instead, the robot receives goals from a human gradually during learning II-C. Formally, a goal is a binary classifier over states $g : \mathcal{X} \mapsto \{0, 1\}$ where 1 indicates that a state is within the goal set. We refer to a combination of an initial state x_0 and goal g as a *task*. *Solving* a task entails taking actions to reach a state x_t where $g(x_t) = 1$ from x_0 within a maximum time-step horizon H_{eval}.

B. Planning to Solve Tasks

Given a task, the robot will plan to generate actions that are likely to accomplish the goal from the initial state. Following previous work [1, 35, 51, 55], we decompose planning into two levels: skill sequencing and continuous parameter selection. Skill sequencing consists of generating a *skeleton*, e.g., (`MoveTo(ball, ∘)`, `Pick(ball, floor, ∘)`, `MoveTo(table, ∘)`, `Place(ball, table, ∘)`). Given a skeleton, we select continuous parameters using *parameter policies* $\theta \sim \pi_u(\cdot \mid x)$. Since parameter selection is condi-

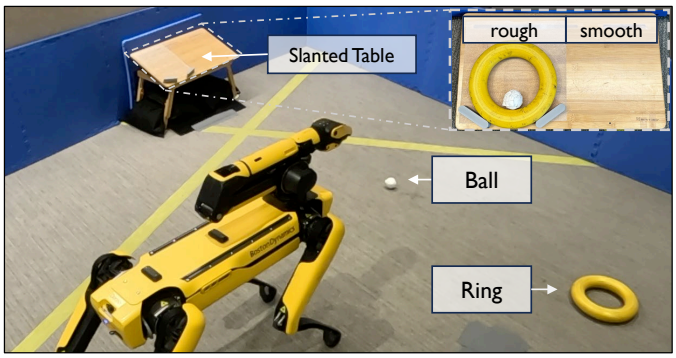

Fig. 2: **Running example: Ball-Ring environment.** The goal is to put the ball on the table. The robot should learn that (1) the ball cannot be placed directly because it will roll off the slanted table; (2) the ring can only be placed on the left side because the right side is smooth (shown in the top-right corner); (3) placing the ring on the table and then placing the ball inside the ring is the best way to accomplish the goal.

Algorithm 1: Planning and Execution

1 **Input:** Current state x and goal g.
2 Generate a skeleton $(u_0, \ldots, u_n)$ to g from x.
3 For $i = 0, \ldots, n$:
4 Sample $\theta \sim \pi_{u_i}$ and **execute** $u_i(\theta)$.
5 Perceive and update the current state x.
6 If $J_{u_i}(x) \neq 1$, repeat from line 2 (replan).

Algorithm 2: Online Learning Paradigm

1 **Initialize** parameter policies $\Pi = \{\pi_u^0 : u \in \mathcal{U}\}$.
2 **Repeat**:
3 **If** a human has given a goal g:
4 Plan and execute to g with Algorithm 1.
5 **Else**:
6 Select and execute actions of the robot's choice.
7 Update Π every m iterations.

tioned on the state x, and since we are not assuming a known transition distribution [51], we sample and execute each skill *greedily*. If the skill terminates and does not meet its success condition, we replan. See Algorithm 1 for a summary.

The robot will learn parameter policies through online experience (Section II-C). We assume that each skill u is accompanied by a *parameter prior* π_u^0 to be used before any parameter policies have been learned. For example, a pick skill may have an associated grasp sampler that provides valid grasps some percentage of the time. At first, the robot uses the parameter priors to select parameters ($\pi_u = \pi_u^0$), but as the robot collects online experience, it will learn to improve the parameter policies with respect to the environment and task distribution. For example, the robot should learn grasp samplers that are specialized to the objects in the environment that need to be manipulated.

How can we generate skeletons to maximize the probability that sampling will succeed? Towards answering this question, we introduce the notion of *skill competence*.

Definition 1 (Skill Competence)**.** The *competence* $c_{u,\pi}$ of a skill u with current parameter policy π_u is the expected success $\mathbb{E}[J_u(X_{t+1}) \mid I_u(X_t)]$, where X_t is a random variable for the state before skill execution, A_t is a r.v. for the action generated from $\theta \sim \pi_u(\cdot \mid X_t)$, and $X_{t+1} \sim P(\cdot \mid X_t, A_t)$.

For example, if `Pick(ball, floor, o)` successfully grasps the ball from the floor 80% of the time, the competence would be 0.8. Note that competence is defined in terms of the current parameter policy, and that the distribution of X_t is induced by the overall planning procedure and the task distribution. In practice, skill competences are unknown and must be estimated from data (Section III).

To establish a relationship between skill competence and full skeleton success, we introduce a strong assumption:

Assumption 1. *Success $J_u(X_{t+1})$ is independent from the state X_t conditioned on the initiation condition $I_u(X_t) = 1$.*

In other words, the success rate of a skill is the same for all states in its initiation set. This assumption has been previously considered in different forms [1, 33], but it does not always hold in practice. For example, the specific grasp of an object may influence the success rate of placing. We can mitigate this by replanning, but to fully remove the assumption, we would need task and motion planning [24, 55] or automated skill partitioning [1, 33], which we leave to future work.

We can now revisit the problem of generating a skeleton that has the maximum likelihood of success. Given Assumption 1, we want to find a skeleton $(u_0, \ldots, u_n)$ with three properties: (1) $\prod_{i=0}^{n} c_i$ is maximal, where $c_0, \ldots, c_n$ are the corresponding competences for the skills in the skeleton; (2) the initiation and success conditions for subsequent skills chain together (see [33] for a formal definition); and (3) the goal is achieved. Following previous work [1, 33, 51, 52], we take advantage of the close relationship between (parameterized) options and AI planning operators to generate skeletons that satisfy conditions (2) and (3). Previous work has considered how to learn these operators automatically; we manually specify them for this work. To satisfy condition (1), we associate a cost of $-\log(c)$ to the respective operator and use an off-the-shelf AI planner [28] to find a minimal cost (maximum likelihood) skeleton. See Appendix A for further details.

C. Online Learning Paradigm

We want the robot to get better at solving tasks over time. We consider a reset-free online learning [25, 38, 58] paradigm where the robot is sometimes given a task to solve and otherwise given *free time* during which it should autonomously learn to improve. The **key question** is: what should the robot do during free time to get better at solving tasks?

We assume that the skills themselves are fixed (e.g., for safety reasons), but the parameter policies can change. The robot should therefore use its free time to improve its parameter policies, specializing the given parameter priors to the particular objects, goals, and constraints in its environment.

Algorithm 3: Planning to Practice

1 **Input**: Current parameter policies Π.
2 Select a skill $u \in \mathcal{U}$ to practice (see Algorithm 4).
3 Plan to I_u using Algorithm 1 with Π.
4 Practice the skill u one time:
5 Sample parameters θ from an *explore policy* π_u^+.
6 **Execute** $u(\theta)$ and record the transition.
7 **Repeat** from line 2 until free time expires.

Algorithm 4: Selecting a Skill to Practice

1 For each $u \in \mathcal{U}$ with current parameter policy π_u:
2 **Estimate** the current competence $c_{u,\pi}$.
3 **Extrapolate**: predict $c_{u,\pi'}$, the competence after practicing u and updating π_u to π'_u.
4 **Situate** the competence in the task distribution, computing $J_{\text{skill}}(u) \triangleq J_{\text{tasks}}(\Pi - \{\pi\} \cup \{\pi'\})$.
5 **Return** $\text{argmax}_u J_{\text{skill}}(u)$ for practice.

This setup is summarized in Algorithm 2. Note that this setup is fully autonomous; the environment is not reset. Our main interest is Line 6: how should the robot choose actions to gather data for improving its parameter policies?

III. Planning to Learn

We propose that the robot should spend its free time *planning to practice* skills. In particular, we commit to the meta-strategy shown in Algorithm 3, where the robot repeatedly selects a skill to practice, plans to satisfy that skill's initiation condition, samples parameters from an *explore parameter policy*, executes the action, and records the result. In using this meta-strategy, we make two assumptions.

Assumption 2. *For $x \in \mathcal{X}$ and $u \in \mathcal{U}$, there exists a sequence of actions that reach I_u from x with nonzero probability.*

In other words, it is not possible to get permanently "stuck" during online learning. This assumption can be weakened if certain skills do not need to be practiced infinitely often.

Assumption 3. *(Informal) Parameter priors have support over good parameter choices.*

For efficiency (and perhaps safety) purposes, we will not permit the robot to sample arbitrarily from skill parameter spaces; we therefore assume that the priors are sufficiently broad to enable learning. Given these assumptions, we are left with three decisions:

1) How should we decide what skills to practice?
2) What explore parameter policies π_u^+ should we use?
3) How should we update the parameter policies?

In this work, we choose to focus on the first question and draw on existing techniques to answer the second two. See Figure 3 for an overview of the full pipeline.

A. Selecting Skills to Practice

Given the relationship between competence and task success, a natural answer to the first question would be to practice the skill with the lowest current competence. However, there there are two major issues with this "Fail Focus" strategy. First, a low-competence skill may be impossible to improve. For example, in the Ball-Ring environment, the `Place(ball, table, o)` skill is bound to fail since the table is slanted (as seen in Figure 2). Second, even if a low-competence skill could be improved, the skill may be less critical for the task distribution than others. In the worst case, Fail Focus may cause the robot to spend all its free time attempting to improve an impossible skill that is of no consequence to any given task.

A better skill selection strategy would be more directly tied to our real objective: to efficiently and effectively solve the tasks given to the robot. We consider a close proxy to this real objective. Given parameter policies $\Pi = \{\pi_u : u \in \mathcal{U}\}$ and a task (x_0, g), let $J_{\text{task}}(\Pi, x_0, g)$ be the probability that planning succeeds without replanning. From Section II-B, we have that $J_{\text{task}}(\Pi, x_0, g) = \prod_{i=0}^{n} c_i$, where c_i is the competence of the i^{th} skill in the skeleton generated for the task. Given a task distribution (X_0, G), our overall objective is to learn parameter policies that maximize:

$$J_{\text{tasks}}(\Pi) \triangleq \mathop{\mathbb{E}}_{X_0,G}[J_{\text{task}}(\Pi, X_0, G)], \tag{1}$$

In other words, the expected probability that planning succeeds without replanning over the task distribution.

We propose to practice the skill whose predicted improvement would maximally increase $J_{\text{tasks}}(\Pi)$. In other words, we will practice **the skill with the greatest expected improvement to the overall distribution of human-given tasks**. We do this in three key steps (Algorithm 4). For each skill, we: (1) *Estimate the competence*: compute the current competence of the skill from data; (2) *Extrapolate the competence*: predict how the competence of the skill would change if it were practiced once more and then its parameter policy was updated; (3) *Situate the competence*: predict how the overall success rate on the task distribution would change given the extrapolated skill competence. This approach resolves the issues with Fail Focus: by extrapolating, we avoid practicing impossible or plateaued skills; and by situating, we avoid practicing irrelevant or unimportant skills. We now describe these steps in detail.

1) Estimating Skill Competence: Our first task is to estimate the current competence of a skill based on the transitions that have been collected thus far. To estimate competence, we propose a graphical model that explicates the relationship between competence, transitions, and learning.

Recall that parameter policies are updated periodically (Algorithm 2); we refer to each period with the same parameter policy (i.e., before an update is made) as a *learning cycle*. Thus, the robot's free time is composed of a series of learning cycles. Let $S_{i,k}$ be a binary random variable for the result of the success condition on the k^{th} usage of the skill in learning

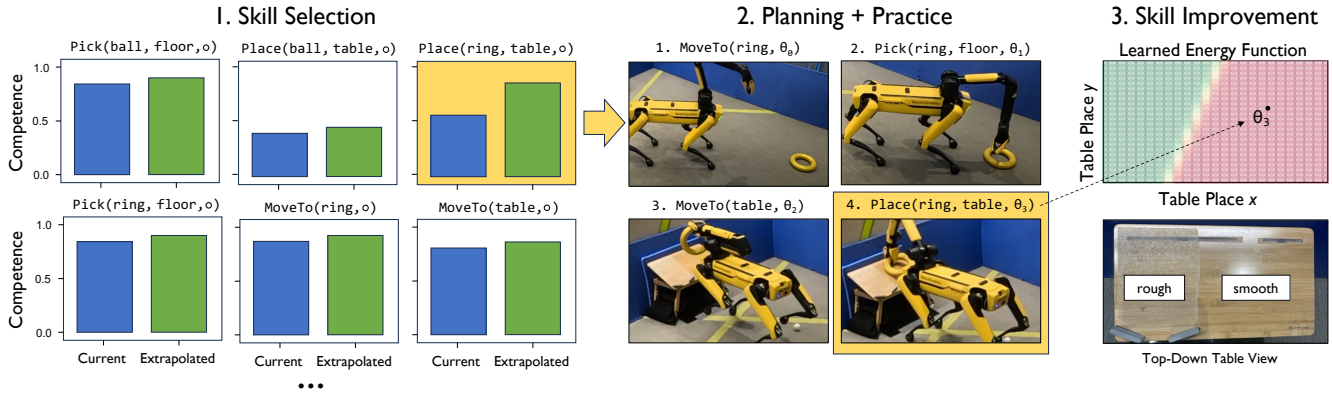

Fig. 3: **Pipeline overview.** (1) During free time, the robot repeatedly selects skills to practice. Here, `Place(ring, table, ○)` is selected because it maximizes J_{skill} (Algorithm 4). (2) The robot plans to satisfy the initiation condition of the skill and then selects a continuous parameter to practice (Algorithm 3). (3) The resulting success or failure of the skill is used to improve the parameter policy (Section III-C).

cycle i.[1] Note that different skills are selected for practice each cycle, so the number of skill usages (maximum k values) varies and can be zero. In practice, this number is low, so we cannot reliably estimate the competence of a skill based on data from the current cycle alone.

Let C_i be a random variable for the skill's competence *during* learning cycle i (before it has re-learned using the data from that cycle). To estimate skill competence, we wish to know $P(C_k \mid S_{1,1}, S_{1,2}, \ldots, S_{1,n_1}, S_{2,1}, \ldots, S_{k,n_k})$, that is, the conditional distribution of competence *now* given all observations up until now. We consider a Bayesian time series model with a joint distribution that factorizes as follows:

$$P(S_{1:n_1}, ..., S_{k:n_k}, ..., C_{1:N_{\text{cycle}}}) = \prod_i P_i(C_i) \prod_k P(S_{i,k} \mid C_i)$$

where $P(S_{i,k} \mid C_i)$ is the *observation model* and $P_i(C_i)$ is a *cycle prior*. For the observation model, we use a Bernoulli:

$$P(S_{i,k} = 1 \mid C_i = c) = c.$$

For the cycle priors, we use Beta distributions, since the Beta is the conjugate prior of the Bernoulli. Let us first consider $P_0(C_0)$, the prior competence before any observations have been made. We define one prior for all skills that has high mean but large variance (in experiments, $\text{Beta}(10, 1)$), reflecting our weakly held expectation that parameter priors π_u^0 will be generally good. This also introduces a form of optimism, which can be helpful for exploration [4, 41].

For subsequent cycle priors $P_i(C_i)$, we assume that a skill's competence is some function of the size of the dataset used to learn that skill's parameter policy. Note that we are *not* positing a *general* relationship between data count and competence. One skill may always have perfect competence; another may always have zero competence; and a third may improve as data increases. Let $f_\phi : \mathbb{Z}_{\geq 0} \to \text{Beta}(\alpha, \beta)$ be a *competence model* where the input is the number of data used for learning, the output is a Beta distribution over competence and ϕ indicates that the function belongs to a hypothesis class $\mathcal{F}$. The cycle prior $P_i(C_i)$ is given by $f_\phi(m_i)$ where m_i is the number of data collected for the skill through cycle i. For example, in Ball-Ring, a good competence model for the `Place(ball, table, ○)` skill would output a near-zero Beta for any input, because no amount of data can improve the skill. On the other hand, a competence model for `Place(ring, table, ○)` should output Beta distributions with increasing modes, since that skill *can* improve with practice.

To estimate the current skill competence, we need to infer C_i for all i and fit ϕ for f_ϕ. We considered two approaches: a principled expectation-maximization (EM) approach, and a much simpler sliding-window-based approach. In preliminary experiments, we found the simpler approach to perform at least as well as EM, and its behavior was much easier to interpret, so we used it for our main experiments. See Appendix B.

2) Extrapolating Skill Competence: Given the competence model f_ϕ fit during estimation, extrapolation is straightforward: we can simply evaluate $f_\phi(m+1)$ to predict how the skill competence would change if we collected one more data point of practice, where m is the number of data seen so far. Let $\hat{c}$ denote the mode of $f_\phi(m+1)$, i.e., the most likely next competence. Here we assume that a skill's competence never gets worse with learning: $\forall \phi$, if $m' > m$, then $\mathbb{E}[f_\phi(m')] \geq \mathbb{E}[f_\phi(m)]$. We can enforce this assumption by choosing $\mathcal{F}$ appropriately. This assumption may not always hold in practice, but for the purpose of extrapolation, it is important that the agent be optimistic and not deliberately avoid collecting additional data for a skill.

3) Situating Skill Competence: Our final step is to predict the expected improvement to the overall task distribution given the extrapolated competence. Let Π' be the set of current parameter policies Π, but with the parameter policy π_u for the current skill under consideration u replaced with π'_u, a hypothetical policy that would result from practicing u once more and re-learning. We wish to compute $J_{\text{tasks}}(\Pi')$ (Equation 1), the expected probability that planning would succeed (without replanning) over the task distribution.

To compute $J_{\text{tasks}}(\Pi', x_0, g)$ for a given task x_0, g, we need not know π'_u itself, but only the competence of π'_u, which

[1] We have not yet defined the explore policy, but it is important here that only "exploit" samples are used to estimate competence; see Section III-B.

we have computed by extrapolating. To complete $J_{\text{tasks}}(\Pi')$, we need to take an expectation over the task distribution. As mentioned in Section II, we do not assume that the robot has direct access to the task distribution; instead, we collect the states and goals used to query the planner, including when replanning is triggered (Algorithm 1) and use that empirical task distribution to approximate $J_{\text{tasks}}(\Pi')$.

B. Explore Parameter Policies

After we have selected a skill to practice and planned to satisfy its initiation condition, we must decide what parameters to use (Algorithm 3). We can view this parameter selection problem as a contextual bandit with infinite arms [6, 37]. Here, the context is the current state x_t, the actions are parameters θ, and the reward is 1 if the success condition passes and 0 otherwise. To balance exploration and exploitation, we use an epsilon-greedy policy:

$$\pi_u^+ = \epsilon\pi_u^0 + (1-\epsilon)\pi_u.$$

Other choices are possible; we use this simple approach to maintain focus on the skill selection problem.

C. Learning to Improve Parameter Policies

To complete our approach for planning to practice parameterized skills, we must now determine how the collected experience can be used to improve the parameter policies. Recall that for each skill u, we are given a parameter prior π_u^0 and we wish to learn an improved parameter policy π_u. Recall also that we have recorded transition data, which we can partition by skill and label according to whether the J check for that particular skill passed: $\mathcal{D}_u = \{((x_t, a_t), J_u(x_{t+1})) : a_t \text{ uses skill } u\}$. Given these data, many approaches are possible. Our approach is to learn an implicit (energy) function $E_u : \mathcal{X} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$ to define the parameter policy:

$$\pi_u(\theta \mid x) \propto \pi_u^0(\theta \mid x) E_u(x, u(\theta)).$$

Specifically, we train small neural network classifiers to minimize binary cross entropy loss and then use the classifier log probabilities for E_u. In contrast to the explore policy, the parameter policy π_u is meant to exploit, so we select parameters via $\text{argmax}_\theta\, \pi_u(\theta \mid x)$. In practice, we sample 100 candidates from the prior $\pi_u^0(\theta \mid x)$ and select the maximum.

For real-robot experiments, it is essential that parameter policies can be learned from very little data. Beyond using good parameter priors, we take two additional steps for data efficiency. First, we share neural network weights between parameter policies that have the same "parent" skill but different object parameters (e.g., `Place(ball, table, o)` and `Place(ring, floor, o)`). Second, we perform feature engineering for learning by mapping the full state and action to a low-dimensional vector that is input to E_u. After applying these features to the data in $\mathcal{D}_u$, we are left with a standard binary classification dataset. See Appendix E for details of the feature mapping as well as the training and use of E_u.

IV. Experiments

Our experiments are designed to empirically answer the following questions about our approach (EES):

Q1. To what extent does EES choose skills for practice that lead to improvements in task distribution success rate, especially compared to alternative approaches?
Q2. How sample efficient is EES compared to alternatives?
Q3. To what extent is EES aware of the task distribution?

Environments. We now provide high-level environment descriptions with details in Appendix F. We use three simulated environments of varying complexity and two real-robot analogs to simulated environments. For details on our real-robot setup, see Appendix D. See also the supplementary material for time-lapse videos of the real robot practicing skills and learning over time.

- *Light Switch (Simulated)*: A toy 1D grid environment. The robot starts in the leftmost room (grid cell) and must switch on a light in the rightmost room. The light is controlled by a dial in the same room that must be precisely actuated. The robot has skills to move left or right, turn the dial to a sampled setting, and also try to "jump" from a particular room all the way to the final room with the dial, though this jump skill is impossible and always fails. We use a grid size of 25 rooms in our main experiments.
- *Ball-Ring (Simulated)*: A simulated version of the environment depicted in Figure 2. To add complexity, the simulated version features multiple tables, some of which are slanted and partially smooth (as shown in the figure) and others that are standard flat surfaces. The relative locations of the smooth patches vary between tables. The robot does not initially know that placing the ball on a slanted surface will fail, nor does it know that placing the ring on the smooth part of a slanted table will fail with high probability. The robot has skills for moving, picking, and placing the ball and ring.
- *Ball-Ring (Real)*: The real version of the previous environment. See Figure 2 and the supplementary video.
- *Cleanup Playroom (Simulated)*: A simulated version of the environment depicted in Figure 1. The robot is tasked with cleaning up a child's playroom by putting two toys into a bin. The toys start out atop a table that may be blocked by a chair. There is also a brush on the floor. The robot again has skills for moving, picking, and placing, but also for grasping and dragging the chair, grasping and dumping out the bin, and sweeping toys from the table into the bin. The success of sweeping depends on the relative positions of the toys and bin and the sweeping velocity.
- *Cleanup Playroom (Real)*: The real version of the simulated Cleanup Playroom environment. See Figure 1 and the supplementary video.

Approaches. We now briefly describe all of the approaches that we evaluate. See Appendix E for details. The first five approaches are alternative instantiations of *planning to practice* (Algorithm 3); the last three do not use that meta-strategy.

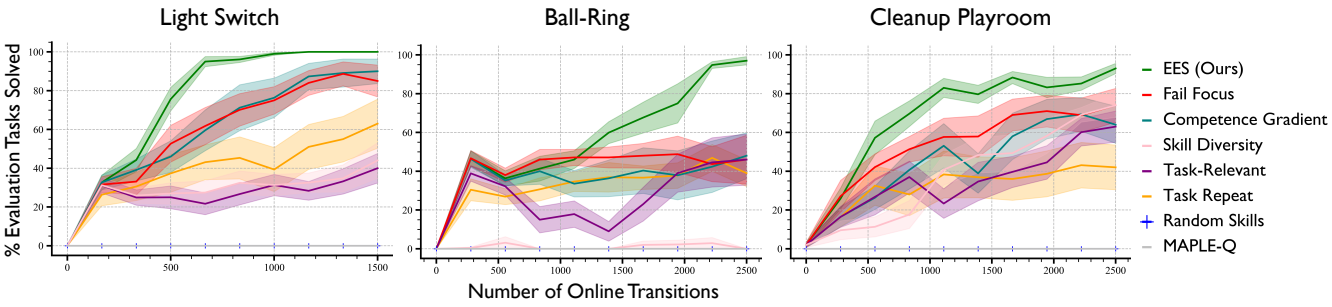

Fig. 4: **Simulation results.** Percentage of evaluation tasks solved vs. number of online transitions collected for all approaches in all simulated environments. Solid lines represent means and shading represents standard error across 10 seeds.

# Steps	S1	S2	S3	S4	S5	Mean
0	0.0	0.0	0.0	0.0	0.0	**0.0**
120	0.0	0.33	0.0	0.67	0.0	**0.2**
240	1.0	0.67	1.0	0.67	0.67	**0.8**

TABLE I: **Ball-Ring real-robot results.** Steps are online transitions; S1 = Seed 1. Entries are fractions of 3 evaluation tasks solved.

# Steps	S1	S2	S3	S4	S5	Mean
0	0.0	0.67	0.33	0.67	1.0	**0.53**
120	0.67	1.0	1.0	1.0	1.0	**0.93**
240	1.0	1.0	0.67	1.0	1.0	**0.93**

TABLE II: **Cleanup Playroom real-robot results.** See Table I caption for details.

- *Estimate, Extrapolate, Situate (EES)*: Our main approach.
- *Fail Focus*: Practices the skill with the lowest current estimated competence.
- *Competence Gradient*: Inspired by [5, 13, 56], practices the skill with the highest expected competence gain (i.e., difference between current and extrapolated competence).
- *Skill Diversity*: Practices the least-practiced skill.
- *Task-Relevant*: Practices a randomly selected skill among those that have been previously included in some maximum-likelihood task plan.
- *Task Repeat*: Samples a task from the empirical task distribution and plans to the task goal. If the task goal is already reached, plans instead to the task initial state.
- *Random Skills*: Repeatedly executes random skills selected from those whose initiation conditions are satisfied in the current state. Does not plan.
- *MAPLE-Q*: Uses hierarchical reinforcement learning to learn to select both skills and parameters. Does not plan. This approach is a modified version of MAPLE [44], which was developed for a more difficult problem setting. Our modifications are meant to give the approach access to the same prior knowledge as the other approaches (e.g., explicit option definitions and parameter priors), but a direct comparison remains challenging. See Appendix E for further discussion and details.

Experimental Setup. For all simulated environments, we run 10 random seeds of each approach. Task generation, parameter prior sampling, tie-breaking during planning, and stochastic environment transitions all vary between seeds. Within each environment, the number of free periods, the number of steps within each free period, and the evaluation horizon H_{eval} is the same for all approaches (see Appendix F). After every free period, each approach is evaluated on 10 randomly sampled "evaluation tasks" with held-out initial states. All simulation experiments were conducted on a quad-core Intel Xeon Platinum 8260 processor with a 192GB RAM limit. For real-robot environments, we run our main approach only (due to the intense time and resource requirements) and use 5 random seeds. Each seed represents a fully independent learning trial. We load model checkpoints after learning and use 3 tasks to evaluate performance. For all experiments, our key quantitative measure is the robot's task success rate under the evaluation horizon.

Results and Analysis. Figure 4 shows a plot of the evaluation success rate of all approaches in all simulated environments. EES is consistently the most sample efficient, achieving higher success rates after fewer online transitions than the baselines. Fail Focus falters not only because it focuses on impossible skills, like jumping in Light Switch or placing the ball on the table in Ball Ring, but also because it is not sufficiently situated in the task distribution. For example, there are no impossible skills in Cleanup Playroom, but EES still outperforms Fail Focus. Competence Gradient is competitive with Fail Focus but similarly lacks situatedness. The poor performance of Skill Diversity, Task-Relevant, Task Repeat, and Random Skills underscore the importance of directed active practice in these long-horizon environments with many possible skills. Like the Random Skills baseline, MAPLE-Q fails to solve any evaluation tasks, which is not surprising given the highly limited number of online transitions that we are considering; previous related work [44] learns from multiple orders of magnitude more data. We verified that our implementation does well in far simpler environments with far more data; see Appendix E for further discussion.

Tables I and II show EES evaluation results for the real-robot environments. Our approach is able to improve its

Goals	`Pick(brush)`	`Drop(toy, bin)`	`Sweep(...)`
Both toys	217.1	0.0	110.2
One toy only	88.4	305.9	0.0

TABLE III: **Skills practiced with varying task distributions.** The entries are the total number of times EES chooses to practice a select number of skills in the simulated Cleanup Playroom environment (averaged over 10 seeds). See text for discussion.

performance after 120 and 240 real-world skill-executions respectively. Each seed took 1-3 hours of real robot time; see the supplementary video. These results are especially noteworthy given the non-trivial noise in perception and control, leading to both false positives and false negatives in the skill datasets ($\mathcal{D}_u$) collected for parameter policy learning.

Towards answering Q3 (to what extent is EES aware of the task distribution?), we conducted an additional experiment in the simulated Cleanup Playroom environment. Rather than giving the robot the goal of putting *both* toys into the bin, we instead gave the goal of putting *one* toy in the bin. As shown in Table III, this task distribution change led to a corresponding change in practicing behavior: rather than practicing sweeping, the robot instead practiced *dropping* the toy into the bin directly. This finding matches intuition: sweeping is unnecessarily complicated when only one object needs to be stowed (a pick-and-place strategy is better), but worthwhile when two objects can be stowed with one sweep.

V. Related Work

A. Exploration in Reinforcement Learning

The problem of sequentially selecting actions that lead to efficient learning is central to *exploration* in reinforcement learning [2, 8, 13, 29, 46]. One important difference between our setting and RL is that we do not have a temporal credit assignment problem: given skill success conditions and Assumption 1, each parameter policy learning problem is self-contained. Our skill selection problem is therefore related to exploration in multi-armed bandits [9], but different still, since selecting a skill to practice does not lead to a task reward, but rather, to a data point that can be used to improve the parameter policy for that skill. Note that the inner problem of choosing parameters for a selected skill *is* a bandit problem (with infinite arms) [10] as explained in Section III-B, but our primary interest is the outer skill selection problem.

Within the RL literature, the most related work to ours is that of Stout and Barto [56], Baranes and Oudeyer [5] and Colas et al. [13], who each consider a form of *competence progress* to guide exploration. Compared to our approach, these previous works *estimate* and *extrapolate* competence, but they do not *situate* the competence in a task distribution. Their motivation is different from ours—they assume that a task distribution is not known and consider the problem of deriving intrinsic motivation in the absence of goals [48]. The Competence Gradient baseline in our experiments is inspired by these works and confirms the importance of situating competence in our setting.

B. Parameter Policy Learning in RL

The problem of learning skill parameter policies has also been considered in the RL literature, for example, in Parameterized Action MDPs (PAMDPs) [15, 27, 40, 44]. In addition to facing the challenge of temporal credit assignment, these works typically do not assume a given method for discrete skill sequencing and instead need to learn a high-level "manager" policy in addition to the "worker" parameter policies. Altogether, this represents a much harder problem setting than ours, and the sample complexity of current techniques remains prohibitively high for the kind of rapid skill specialization we consider here (for example, see our MAPLE-Q baseline [44]). Assuming that the environment can be automatically reset is the norm in this literature, with some notable exceptions [25, 38, 58]. Work by Ames et al. [1] is a step toward bridging the RL setting and our setting; they automatically derive AI planning operators from parameterized option specifications. Exploration is not a central consideration in these works (but see [3, 14]).

Recent work in RL also considers specializing (fine tuning) skills through online learning [26, 62], starting from generic (pre-trained) distributions. Other recent work by Fang et al. [19] is another example of *planning to practice*. Their planning uses a learned latent subgoal space, rather than options and AI planners. They learn goal-conditioned policies that are analogous to our parameter policies. Active learning at the skill level is not a primary focus; their approach is similar to our Task Repeat baseline (but with environment resets instead of reset-free learning). Follow-up work [20] considers active learning more centrally and proposes a method for generating subgoals for online learning using a diversity-based metric.

C. Learning Samplers for Task and Motion Planning

In the context of the task and motion planning (TAMP) literature [24], our parameter policies can be seen as *samplers* for refining skeletons generated by task planning. TAMP approaches typically do not make Assumption 1 and instead sample parameters contingent on the entire skeleton (e.g., selecting grasp parameters that enable future constrained placements).

Several works have considered learning samplers for TAMP [12, 30]. Silver et al. [52] learn samplers from an offline demonstration dataset. The details of our neural-network learning over object-centric states are most similar to theirs. Other recent work has considered learning samplers with diffusion models [43, 64]. Most relevant of these is the work by Mendez-Mendez et al. [42], who consider diffusion-based sampler learning for TAMP in an *embodied lifelong* setting. However, in that work, the robot is not given free time; it remains in task time throughout online learning. Additionally, the agent does not have separate exploration and exploitation samplers, but rather only an exploitation sampler. Wang et al. [60] consider active sampler learning for TAMP with a focus on the inner bandit problem of selecting parameters to practice for a given skill. In principle, their parameter selection method could be swapped in for our epsilon-greedy approach.

In the TAMP literature, the work by Noseworthy et al. [45] is another instance of active learning. They learn to predict whether a skeleton is feasible [18, 61, 63], i.e., whether there exists continuous parameters that would achieve the goal (typically in a deterministic setting). Future work could combine active feasibility prediction with our active parameter policy learning as a path toward removing Assumption 1 and scaling to more geometrically complex environments.

VI. Limitations and Future Work

In this work, we proposed Estimate, Extrapolate & Situate (EES) as a method for planning to practice parameterized skills. We found that simulated and real robots using EES are able to rapidly and continually improve their parameter policies with respect to human-given task distributions. Our real-robot results are particularly noteworthy as instances of reset-free online learning in challenging, long-horizon mobile manipulation environments.

There are several limitations of the present work and of EES as a general method. For the sake of rapidly learning on a real robot, we started with a considerable amount of prior knowledge: known object (feature) detectors, fully-specified parameterized skills (and operators for planning), low-dimensional feature selectors for parameter policy training, and good parameter priors. Previous work has considered learning each of these components (e.g., [52, 54]), but doing so may require significantly more data than what we considered here. We also made Assumptions 1-3, which are strong, and while we need not satisfy them completely to attain good performance, they remain worthy of further scrutiny. Our relatively naive treatment of noise and our assumption of full observability are also clearly limiting. Furthermore, our commitment to *planning to practice* (Algorithm 3) is perhaps overly myopic: better strategies might anticipate that practicing one skill enables quickly practicing another, reasoning over *sequences* of practice attempts. Finally, this work presupposes that robots *should* be practicing and learning during deployment. The extent to which this is true depends greatly on the nature of the deployment and the constraints under which the robot is allowed to practice.

One future direction that could address multiple limitations simultaneously would be to give the agent access to a simulator. Even if the simulator were a coarse approximation of the real world, the robot could nonetheless use it to bootstrap real-world practice time. The same simulator could be used for reasoning about potentially irreversible actions before executing them in the real world, and for integrated task and motion planning (TAMP) [24] towards removing Assumption 1. Leveraging TAMP would also be a step toward more principled planning in stochastic [50] and partially-observable environments [23].

Acknowledgments

We gratefully acknowledge support from NSF grant 2214177; from AFOSR grant FA9550-22-1-0249; from ONR MURI grant N00014-22-1-2740; from ARO grant W911NF-23-1-0034 and from the MIT Quest for Intelligence. Nishanth, Tom, and Willie are supported by NSF GRFP fellowships. We thank Will Shen for feedback and suggestions on an early paper draft, as well as invaluable help with creating the accompanying website. We also thank Russell Mendonca for helpful early discussions, especially with respect to the MAPLE-Q baseline. We are grateful for helpful discussion, brainstorming, and support from Stefanie Tellex, Ashay Athalye, Tushar Kusnur, Jiuguang Wang, Gustavo Goretkin, Andrew Messing, Joe St. Germain and others at the AI Institute. We thank Hannah Blumberg for invaluable early help with designing one of our robot domains, as well as helpful comments on an early draft of this paper. We acknowledge the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing HPC resources that have contributed to the simulation results reported within this paper. Finally, we wish to thank our three Spot robots, Moana, Donner, and Kepler, for being so reliable throughout the extensive prototyping and experimentation required for this paper. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

References

[1] Barrett Ames, Allison Thackston, and George Konidaris. Learning symbolic representations for planning with parameterized skills. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8594313.

[2] Susan Amin, Maziar Gomrokchi, Harsh Satija, Herke van Hoof, and Doina Precup. A survey of exploration methods in reinforcement learning. *arXiv preprint arXiv:2109.00157*, 2021. URL https://arxiv.org/pdf/2109.00157.pdf.

[3] Garrett Andersen and George Konidaris. Active exploration for learning symbolic representations. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. URL https://dl.acm.org/doi/pdf/10.5555/3295222.3295254.

[4] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research (JMLR)*, 2002. URL https://www.jmlr.org/papers/volume3/auer02a/auer02a.pdf.

[5] Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 2013. URL http://www.pyoudeyer.com/ActiveGoalExploration-RAS-2013.pdf.

[6] Donald A. Berry and Bert Fristedt. Bandit problems. sequential allocation of experiments. *Monographs on Statistics and Applied Probability.*, 1987. URL https://onlinelibrary.wiley.com/doi/epdf/10.1002/bimj.4710290105.

[7] Ondrej Biza, Dian Wang, Robert Platt, Jan-Willem van de

Meent, and Lawson LS Wong. Action priors for large action spaces in robotics. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2021. URL https://arxiv.org/pdf/2101.04178.pdf.

[8] Nicolas Bougie and Ryutaro Ichise. Skill-based curiosity for intrinsically motivated reinforcement learning. *Machine Learning*, 109, 2020. URL https://link.springer.com/content/pdf/10.1007/s10994-019-05845-8.pdf.

[9] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in multi-armed bandits problems. In *Algorithmic Learning Theory: 20th International Conference (ALT)*. Springer, 2009. URL http://sbubeck.com/ALT09_BMS.pdf.

[10] Alexandra Carpentier and Michal Valko. Simple regret for infinitely many armed bandits. In *International Conference on Machine Learning (ICML)*, 2015. URL http://proceedings.mlr.press/v37/carpentier15.pdf.

[11] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems (RSS)*, 2023. URL https://diffusion-policy.cs.columbia.edu/diffusion_policy_2023.pdf.

[12] Rohan Chitnis, Dylan Hadfield-Menell, Abhishek Gupta, Siddharth Srivastava, Edward Groshev, Christopher Lin, and Pieter Abbeel. Guided search for task and motion plans using learned heuristics. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016. URL https://people.eecs.berkeley.edu/~pabbeel/papers/2016-ICRA-tamp-learning.pdf.

[13] Cédric Colas, Pierre Fournier, Mohamed Chetouani, Olivier Sigaud, and Pierre-Yves Oudeyer. Curious: intrinsically motivated modular multi-goal reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2019. URL https://proceedings.mlr.press/v97/colas19a/colas19a.pdf.

[14] Bruno Da Silva, George Konidaris, and Andrew Barto. Active learning of parameterized skills. In *International Conference on Machine Learning (ICML)*, 2014. URL https://proceedings.mlr.press/v32/silva14.html.

[15] Murtaza Dalal, Deepak Pathak, and Russ R Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL https://proceedings.neurips.cc/paper/2021/file/b6846b0186a035fcc76b1b1d26fd42fa-Paper.pdf.

[16] Christian Daniel, Malte Viering, Jan Metz, Oliver Kroemer, and Jan Peters. Active reward learning. In *Robotics: Science and Systems (RSS)*, 2014. URL https://www.roboticsproceedings.org/rss10/p31.pdf.

[17] Carlos Diuk, Andre Cohen, and Michael L Littman. An object-oriented representation for efficient reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2008. URL https://carlosdiuk.github.io/papers/OORL.pdf.

[18] Danny Driess, Jung-Su Ha, and Marc Toussaint. Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image. In *Robotics: Science and Systems (RSS)*, 2020. URL https://www.roboticsproceedings.org/rss16/p003.pdf.

[19] Kuan Fang, Patrick Yin, Ashvin Nair, and Sergey Levine. Planning to practice: Efficient online fine-tuning by composing goals in latent space. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022. URL https://arxiv.org/pdf/2205.08129.pdf.

[20] Kuan Fang, Toki Migimatsu, Ajay Mandlekar, Li Fei-Fei, and Jeannette Bohg. Active task randomization: Learning robust skills via unsupervised generation of diverse and feasible tasks. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023. URL https://arxiv.org/pdf/2211.06134.pdf.

[21] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, 2003. URL https://arxiv.org/pdf/1106.4561.pdf.

[22] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. In *arXiv*, 2024. URL https://mobile-aloha.github.io/resources/mobile-aloha.pdf.

[23] Caelan Reed Garrett, Chris Paxton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. Online replanning in belief space for partially observable task and motion problems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5678–5684. IEEE, 2020. URL https://arxiv.org/pdf/1911.04577.pdf.

[24] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 2021. URL https://www.annualreviews.org/doi/pdf/10.1146/annurev-control-091420-084139.

[25] Abhishek Gupta, Justin Yu, Tony Z Zhao, Vikash Kumar, Aaron Rovinsky, Kelvin Xu, Thomas Devlin, and Sergey Levine. Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021. URL https://arxiv.org/pdf/2104.11203.pdf.

[26] Abhishek Gupta, Corey Lynch, Brandon Kinman, Garrett Peake, Sergey Levine, and Karol Hausman. Bootstrapped autonomous practicing via multi-task reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023. URL https://arxiv.org/pdf/2203.15755.pdf.

[27] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. In *International Conference on Learning Representations (ICLR)*, 2016. URL https://www.cs.utexas.edu/users/pstone/Papers/bib2html-links/ICLR16-hausknecht.pdf.

[28] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research (JAIR)*,

2006. URL https://www.jair.org/index.php/jair/article/view/10457/25068.

[29] Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 2002. URL https://www.cis.upenn.edu/~mkearns/papers/KearnsSinghE3.pdf.

[30] Beomjoon Kim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Guiding search in continuous state-action spaces by learning an action sampler from off-target search experience. In *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*, 2018. URL https://ojs.aaai.org/index.php/AAAI/article/view/12106/11965.

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

[32] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023. URL https://arxiv.org/pdf/2304.02643.pdf.

[33] George Konidaris, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 2018. URL https://jair.org/index.php/jair/article/view/11175/26380.

[34] Nishanth Kumar, Willie McClinton, Rohan Chitnis, Tom Silver, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning efficient abstract planning models that choose what to predict. In *Conference on Robot Learning (CoRL)*, 2023. URL https://openreview.net/pdf?id=_gZLyRGGuo.

[35] Nishanth Kumar, Willie McClinton, Kathryn Le, , and Tom Silver. Bilevel planning for robots: An illustrated introduction, 2023. URL https://lis.csail.mit.edu/bilevel-planning-for-robots-an-illustrated-introduction.

[36] Amber Li and Tom Silver. Embodied active learning of relational state abstractions for bilevel planning. In *Conference on Lifelong Learning Agents (CoLLAs)*, 2023. URL https://arxiv.org/pdf/2303.04912.pdf.

[37] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *International Conference on World Wide Web*, 2010. URL https://arxiv.org/pdf/1003.0146.pdf.

[38] Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Reset-free lifelong learning with skill-space planning. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/pdf?id=HIGSa_3kOx3.

[39] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023. URL https://arxiv.org/pdf/2310.12931.pdf.

[40] Warwick Masson, Pravesh Ranchod, and George Konidaris. Reinforcement learning with parameterized actions. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016. URL https://ojs.aaai.org/index.php/AAAI/article/view/10226/10085.

[41] Benedict C May, Nathan Korda, Anthony Lee, and David S Leslie. Optimistic bayesian sampling in contextual-bandit problems. *Journal of Machine Learning Research (JMLR)*, 2012. URL https://www.jmlr.org/papers/volume13/may12a/may12a.pdf.

[42] Jorge Mendez-Mendez, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Embodied lifelong learning for task and motion planning. In *Conference on Robot Learning (CoRL)*, 2023. URL https://openreview.net/pdf?id=ZFjgfJb_5c.

[43] Utkarsh Aashu Mishra, Shangjie Xue, Yongxin Chen, and Danfei Xu. Generative skill chaining: Long-horizon skill planning with diffusion models. In *Conference on Robot Learning*, 2023. URL https://openreview.net/pdf?id=HtJE9ly5dT.

[44] Soroush Nasiriany, Huihan Liu, and Yuke Zhu. Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2022. URL https://arxiv.org/pdf/2110.03655.pdf.

[45] Michael Noseworthy, Isaiah Brand, Caris Moses, Sebastian Castro, Leslie Kaelbling, Tomás Lozano-Pérez, and Nicholas Roy. Active learning of abstract plan feasibility. In *Robotics: Science and Systems (RSS)*, 2021. URL https://www.roboticsproceedings.org/rss17/p043.pdf.

[46] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International Conference on Machine Learning (ICML)*, 2019. URL http://proceedings.mlr.press/v97/pathak19a/pathak19a.pdf.

[47] Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on Robot Learning (CoRL)*, 2021. URL https://arxiv.org/pdf/2010.11944.pdf.

[48] Vieri Giuliano Santucci, Pierre-Yves Oudeyer, Andrew Barto, and Gianluca Baldassarre. Intrinsically motivated open-ended learning in autonomous robots, 2020. URL https://www.frontiersin.org/articles/10.3389/fnbot.2019.00115/full.

[49] Burr Settles. From theories to queries: Active learning in practice. In *Active learning and experimental design workshop in conjunction with AISTATS 2010*, 2011. URL https://proceedings.mlr.press/v16/settles11a.html.

[50] Naman Shah, Deepak Kala Vasudevan, Kislay Kumar, Pranav Kamojjhala, and Siddharth Srivastava. Anytime integrated task and motion policies for stochastic environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9285–9291. IEEE, 2020. URL https://aair-lab.github.io/Projects/STAMP/skkks_icra2020_full.pdf.

[51] Tom Silver, Rohan Chitnis, Joshua Tenenbaum,

Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning symbolic operators for task and motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021. URL https://arxiv.org/pdf/2103.00589.pdf.

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

[53] Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Kaelbling, and Joshua B Tenenbaum. Predicate invention for bilevel planning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2023. URL https://ojs.aaai.org/index.php/AAAI/article/view/26429/26201.

[54] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=Ysuv-WOFeKR.

[55] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *IEEE international conference on robotics and automation (ICRA)*, 2014. URL https://people.eecs.berkeley.edu/~russell/papers/icra14-planrob.pdf.

[56] Andrew Stout and Andrew G Barto. Competence progress intrinsic motivation. In *IEEE 9th International Conference on Development and Learning (ICDL)*, 2010. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1e9521d28184a344c077edaf780c1205b3e90139.

[57] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 1999. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521/pdf?md5=780c0bdb220bb0fa2d0721720296922c&pid=1-s2.0-S0004370299000521-main.pdf.

[58] Sebastian Thrun. A lifelong learning perspective for mobile robot control. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 1995. URL https://www.ri.cmu.edu/pub_files/pub1/thrun_sebastian_1995_3/thrun_sebastian_1995_3.pdf.

[59] Weikang Wan, Yifeng Zhu, Rutav Shah, and Yuke Zhu. Lotus: Continual imitation learning for robot manipulation through unsupervised skill discovery. *arXiv preprint arXiv:2311.02058*, 2023. URL https://arxiv.org/pdf/2311.02058.pdf.

[60] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning compositional models of robot skills for task and motion planning. *The International Journal of Robotics Research IJRR)*, 2021. URL https://arxiv.org/pdf/2006.06444.pdf.

[61] Andrew M Wells, Neil T Dantam, Anshumali Shrivastava, and Lydia E Kavraki. Learning feasibility for task and motion planning in tabletop environments. *IEEE Robotics and Automation Letters (RAL)*, 2019. URL https://europepmc.org/backend/ptpmcrender.fcgi?accid=PMC6491048&blobtype=pdf.

[62] Haoyu Xiong, Russell Mendonca, Kenneth Shaw, and Deepak Pathak. Adaptive mobile manipulation for articulated objects in the open world. *arXiv preprint arXiv:2401.14403*, 2024. URL https://open-world-mobilemanip.github.io/paper.pdf.

[63] Lei Xu, Tianyu Ren, Georgia Chalvatzaki, and Jan Peters. Accelerating integrated task and motion planning with neural feasibility checking. *arXiv preprint arXiv:2203.10568*, 2022. URL https://arxiv.org/pdf/2203.10568.pdf.

[64] Zhutian Yang, Jiayuan Mao, Yilun Du, Jiajun Wu, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Compositional Diffusion-Based Continuous Constraint Solvers. In *Conference on Robot Learning (CoRL)*, 2023. URL https://arxiv.org/pdf/2309.00966.pdf.

[65] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Robotics: Science and Systems (RSS)*, 2023. URL https://arxiv.org/pdf/2304.13705.pdf.

[66] Xingyi Zhou, Rohit Girdhar, Armand Joulin, Philipp Krähenbühl, and Ishan Misra. Detecting twenty-thousand classes using image-level supervision. In *European Conference on Computer Vision*, pages 350–368. Springer, 2022. URL https://github.com/facebookresearch/Detic.

APPENDIX

A. AI Planning Details

In this section, we provide additional details about AI planning to supplement the overview given in Section II-B. We refer the reader to other references [28] for a formal treatment of the planning techniques we use in this work and give a brief overview of the salient points here. First, we assume access to a set of *predicates* and a function `abstract` that maps states to sets of ground predicates. For example, `On` and `HandEmpty` are two predicates in the Ball-Ring environment, and $\texttt{abstract}(x_t)$ could be $\{\texttt{On(ball, floor)}, \texttt{HandEmpty(robot)}, \ldots\}$. We additionally assume that each goal g is associated with a set of ground predicates, e.g., $\{\texttt{On(ball, table)}\}$, which is a subset of $\texttt{abstract}(x_t)$ if and only if $g(x_t) = 1$.

Next, for each parameterized skill, we assume access to a PDDL [21] planning operator with predicate-based preconditions and effects. For example, the operator for `Place(ball, table, o)` is:

```
Place(ball table)
:precondition (and
  (Holding ball)
  (Reachable table))
:effect (and
  (On ball table)
  (HandEmpty)
  (not (Holding ball)))
```

Note the absence of continuous parameters. The operator preconditions characterize the initiation condition of the skill. For example, for the skill above, $I(x) = 1$ if $\{\texttt{Holding(ball)}, \texttt{Reachable(table)}\} \subseteq s$ where $s = \texttt{abstract}(x)$. Similarly, the effects characterize the success condition of the skill. Continuing the example, $J(x') = 1$ if $\{\texttt{On(ball, table)}, \texttt{HandEmpty()}\} \subseteq s'$ and $\{\texttt{On(ball, table)}, \texttt{Holding(ball)}\} \not\subseteq s'$ where $s' = \texttt{abstract}(x')$. Previous work [34, 51, 53] has learned operators and predicates; we manually specify them here.

Given an initial state x_0 and a goal g, we construct a PDDL planning problem with initial state $\texttt{abstract}(x_0)$ and use a PDDL planner [28] to efficiently generate a sequence of planning operators (a skeleton) that chain together to reach the goal. To incorporate skill competences, we associate a cost of $-\log(c)$ to a skill with competence c and find a minimum-cost (maximum-likelihood) skeleton. In experiments, we use LM-Cut (alias `seq-opt-lmcut` in Fast Downward) for minimum-cost planning. We use a planning timeout of 10 seconds. In experiments, this timeout was never triggered.

As explained in Section II-B, once a skeleton is obtained, we greedily select continuous parameters using parameter policies and execute the resulting action. Re-planning is triggered when the success condition (as defined by the operator effects) fails. During evaluation, the robot continues planning and executing until a maximum number of actions H_{eval} is reached (see Appendix F).

B. Competence Models

We now provide details on the competence estimation and extrapolation methods described in Section III-A. Recall that each skill's competence is estimated and extrapolated independently. In the main paper, we presented a graphical model relating the competence, success condition observations, and learning cycles of a skill; that model is summarized Figure 5.

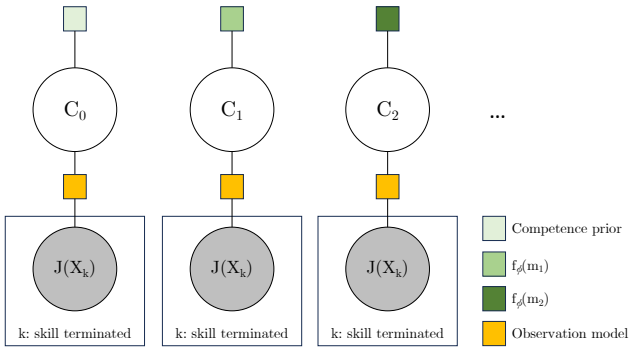

Fig. 5: **Skill competence graphical model.**

As described in the main text, the model components include:

- The observation model $P(J(X_{i,k}) = 1 \mid C_i = c) = c$.
- The initial cycle prior $P_0(C_0)$; we use $\text{Beta}(10, 1)$.
- The other cycle priors $P_i(C_i) = f_\phi(m_i)$ where m_i is the number of data collected for the skill through cycle i and f_ϕ is a learned function that outputs Beta distributions.

We now discuss $f_\phi : \mathbb{Z}_{\geq 0} \to \text{Beta}(\alpha, \beta)$, the *competence model*, in detail. As mentioned, we considered two approaches for learning competence models: one based on expectation-maximization and another based on a simple sliding window. After preliminary analysis, we opted to use the latter in our main experiments, but we describe both here for reference.

Approach 1: Expectation-Maximization. The model in Figure 5 is a latent variable model, so EM is a natural choice. The E step is straightforward: since we have a collection of Beta-Bernoulli distributions, there is a closed-form solution to infer MAP competences given fixed competence models. The M step—fitting the competence models given the most recent MAP competences—is more involved.

To start, we need to find a model class $\mathcal{F}$ with the property that any model in the class is non-decreasing in terms of the modes of the Beta distributions output by the model. (Recall that this property is desirable because we do not want the robot to ever predict that practicing a skill will cause that skill to get worse; that skill would never be practiced.) We use two ideas to satisfy this property. First, rather than having our model output the Beta distribution parameters (α and β) directly, we output the mode and variance and then use those to derive the parameters. Second, for predicting the mode, we pick a model class that is non-decreasing (and bounded between 0 and 1). In our preliminary experiments, we used the exponential function

$$f_\phi(m) = \phi_0 + (\phi_1 - \phi_0)(1 - \exp(-\phi_2 m))$$

where $\phi = [\phi_0, \phi_1, \phi_2]$, $0 \leq \phi_0 \leq 1, \phi_0 \leq \phi_1 \leq 1$, and

$\phi_2 > 0$. Note that $f_\phi(m) = \phi_0$, $\lim_{m\to\infty} f_\phi(m) = \phi_1$, and ϕ_2 controls the rate of increase.

To fit the model, we minimize a loss function $L(\phi) = \sum_i \ell_\phi(\hat{c}_i)$ where $\hat{c}_i$ is the MAP competence for cycle i and $\ell_\phi : [0,1] \to \mathbb{R}$ is the negative log likelihood under f_ϕ. Many techniques are possible to find $\phi^* = \text{argmin}_\phi L(\phi)$, especially since ϕ is only three-dimensional. In preliminary experiments, we used `scipy.optimize`. The results of five iterations of EM on illustrative examples are shown in Figure 6. The performance is good in these examples, but EM is less stable in more realistic cases, which motivates the next method.

Approach 2: Simple Sliding Window. For the experiments in the main paper, we use a much simpler competence model that looks at the recent history of changes in competence and optimistically predicts that the best previous increase in competence between learning cycles will be repeated on subsequent cycles. Concretely, $f_\phi(m+1)$ outputs a Beta with mode $\hat{c}_m + \max_{m-w\le i,j<m}(\hat{c}_j - \hat{c}_i)$, with clipping to enforce $f_\phi(m+1) \le 1$, where $w = 2$ is the window size. (The variance of the Beta distribution is not used during extrapolation.) This approach is quite naive, especially considering that the number of data in each learning cycle is not directly taken into account. Nonetheless, the approach worked well in experiments.

C. Learning Parameter Policy Details

Recall from Section III-C that we learn an energy function $E_u : \mathcal{X} \times \mathcal{A} \to \mathbb{R}_{\ge 0}$ for each skill given a dataset of skill executions (including continuous parameters θ) and success or failure of the J_u check: $\mathcal{D}_u = \{((x_t, a_t), J_u(x_{t+1})) : a_t \text{ uses skill } u\}$. We do this by training a two-layer Multi-Layer Perceptron (MLP) with hidden layers of size $32, 32$ with Binary Cross Entropy (BCE) loss. We use the Adam [31] optimizer with a learning rate of 10^{-3} for 10000 iterations, or until 5000 iterations have passed without any loss change.

As mentioned in the main text, we (1) share weights between skills with the same "parent" and (2) we construct a low-dimensional feature space to facilitate rapid learning. To accomplish (1), we include object IDs in the low-level features and train one neural network per parent. For example, for `Place`, we include features like $[0, 1]$ and $[0, 2]$ to distinguish `Place(ball, table, o)` from `Place(ball, floor, o)`, where 0 is the ID for `ball`, etc. For the rest of the features (2), we default to including the complete set of features for all objects that are included in the skill object parameters. For certain skills that we know to be important for our experiments, we design the feature spaces more carefully:

- In Ball-Ring, for `Place(ring, table, o)`, we include (1) the table size; (2) the x position of the rough patch; (3) the y position of the rough patch; (4) the size of the rough patch; (5) the x position of the table; (6) the y position of the table; and (7) the 2D continuous skill parameters themselves, which represent a relative placement on the table.
- In Cleanup Playroom, for all `Sweep` skills, we include the x and y positions of the object being swept, the x and y position of the bin, and the 1D continuous skill parameter, which represents a sweeping velocity.
- For all `Drop` skills in Cleanup Playroom, we use the 2D parameters alone, which represent a xy position for the gripper relative to the container before the gripper opens.
- For all `Pick` skills in the simulated version of Cleanup Playroom, we use the 2D continuous skill parameters

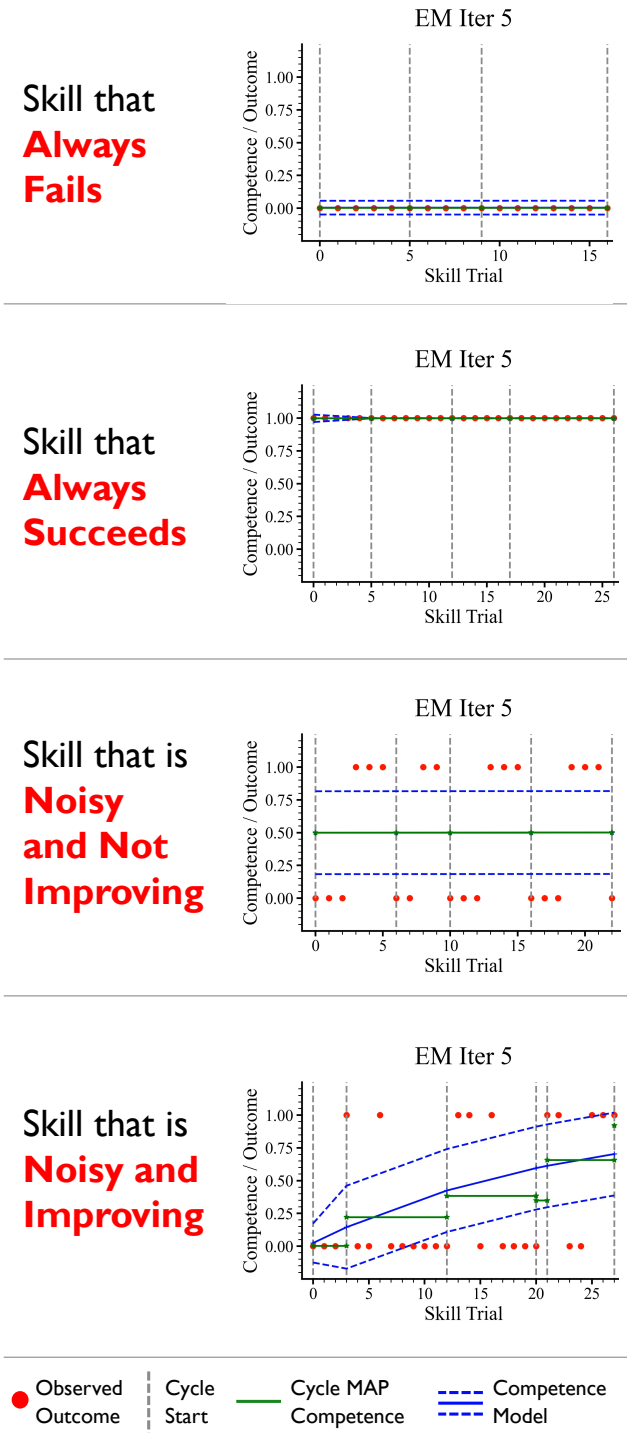

Fig. 6: **Fitting competence models with EM.** For the competence models, solid lines are modes are dashed lines are variances.

alone, which represent a pixel in a canonical view of the object being picked. See Appendix F for further context.

D. Real Robot System Implementation Details

In this section, we provide system details for our real-robot experiments. Recall that our model of the world (Section II-A) comprises object-centric states and parameterized actions. To actualize this model, we need a *perception system* that constructs object-centric states from sensors and *skills* that can be executed on the robot. All real-robot experiments use the Boston Dynamics Spot robot with an arm; see https://bostondynamics.com/products/spot/ for specifications.

1) Real-Robot Perception System: Our perception system has three major components: (1) localization, (2) object detection; and (3) lost object search.

Localization. To implement all our movement skills in a consistent manner that persists between runs, we assume the robot has access to a pre-defined map of its environment. We construct this map by leveraging the Simultaneous Localization and Mapping (SLAM) stack that is part of the Boston Dynamics Spot SDK (see https://dev.bostondynamics.com/docs/concepts/autonomy/graphnav_map_structure). This currently requires placing a number of fiducials around the environment. Given a pre-defined map (which defines a coordinate system centered at the point where the map recording was begun), we implement a `localize()` method that gives us the current location of the robot within the map. We leverage this functionality not only for movement (discussed in the skills subsection below), but also to compute the positions of objects we see in the world, since they are detected relative to the robot's cameras (discussed in the object detection subsection below). The position of the robot itself is also added to the object-centric state. Within the map, we also define the boundaries of all "allowed" regions where the robot can navigate to (disallowed regions include obstacles not captured during mapping such as clear glass walls, etc.). When navigating, we only allow the robot to move to a position that is within the convex hull of points that define the periphery of an allowed region. Additionally, if we detect an object to not be inside an allowed region, we automatically consider that detection invalid and throw it away.

Object Detection. The Spot robot collects RGBD images from six perspectives: one in the hand, two on the front, one on each side, and one on the back. We collect all six images at each time step and then run object detection in each image.

To detect objects in an RGBD image, we use a combination of Detic [66] and Segment Anything (SAM) [32]. See Figure 7 for a summary. Detic uses CLIP embeddings to identify object bounding boxes in RGB images given natural language class names ("prompts"). We experimented with a number of objects and prompts to find a combination that would work reliably in our setting, and even so, significant noise remains. See Table IV for the final set of prompts used. For each bounding box returned by Detic, we run SAM inside the bounding box to get a mask for the object. The center of the bounding box gives the xy position of the object in the camera frame. We then compute the median depth value in the mask to get a z value, and transform the full xyz position into the world frame using the known camera intrinsics and robot pose from localization. The rotation of the object is not detected. Known object features (such as object size, whether it is movable, etc.) are added to the xyz position features. Static objects (e.g., tables) are added to the state automatically, rather than visually detected, for simplicity. As a method for object detection, this overall approach has a number of limitations (see "failure modes" below). However, one advantage is that it is fast—about 0.25 seconds overall—which is important since we are running it at every time step between skill executions.

When the same object is detected in multiple cameras at the same time step, we use the detection with the highest confidence score returned by Detic. Object detections are aggregated over time: whenever an object is detected, the previous detection is overwritten. More sophisticated state estimation strategies are possible. To initialize object detections at the very beginning of online learning, the robot navigates to a fixed home pose, raises its arm to get a top-down view, and rotates in place, collecting images and detecting objects until all known objects have been seen.

Lost Object Search. We assume that all objects can be found in the initial scene by the simple rotate-in-place procedure described above, and that objects are not removed from the scene while the robot is running. Even so, objects can become lost, and it is important for both learning and planning that the robot can find them. A principled approach for object search under partial observability is outside the scope of this paper. Instead, we use the following domain-specific logic. First, when a pick skill is executed, if the robot's gripper is subsequently open beyond a threshold value, then we assume that the target object was successfully grasped and we update a corresponding *held* feature in the object-centric state. Then, if a place or drop skill is executed, and if the held object can be subsequently seen, the held feature is updated accordingly. If a pick skill is executed and the gripper is subsequently closed, or if a place/drop skill is executed and the object is not subsequently seen, the object is declared *lost*. When an object is lost, the robot executes a special *find objects* procedure. This procedure starts by executing a series of fixed move-and-look actions and then begins to randomly sample move-and-look actions until the lost object is seen. In rare cases where more than 10 move-and-look actions are executed and the object is still not found, we manually take control of the robot and point it to look at the lost object. The *find objects* procedure is executed externally from the approach; the transitions collected are not used for learning.

2) Real-Robot Skills: We implement skills for the Spot robot on top of the Boston Dynamics SDK (https://dev.bostondynamics.com/). All skills are listed in Appendix F and implementations are given in the code accompanying the paper (see https://planning-to-practice.github.io). We discuss two types of skills here—*move* and *pick*—and refer to the code for others.

Move skills. The Spot SDK provides functionality for mov-

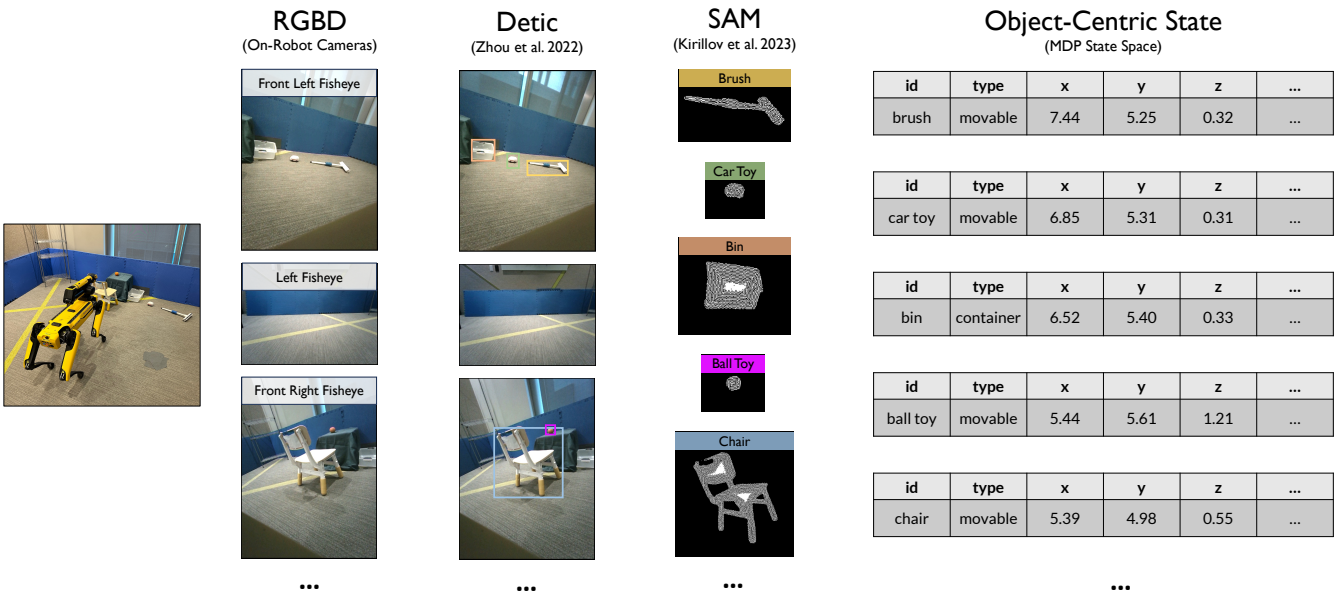

Fig. 7: **Overview of perception pipeline.** We take RGBD images from Spot's cameras and then use Detic [66] and SAM [32] to construct an object-centric state. See text for details.

Environment	Object	Prompt
Ball-Ring	ball	small white ball / ping-pong ball / snowball / cotton ball / white button
Ball-Ring	ring	yellow hoop toy / yellow donut
Cleanup Playroom	bin	white plastic container with black handles / white plastic tray with black handles / white plastic bowl / white storage bin with black handles
Cleanup Playroom	ball toy	small orange basketball / small orange
Cleanup Playroom	car toy	small white ambulance toy / car_(automobile) toy / egg
Cleanup Playroom	platform	black coffee table / bench
Cleanup Playroom	chair	chair
Cleanup Playroom	brush	scrubbing brush / hammer / mop / giant white toothbrush

TABLE IV: **Detic prompts.** The backslashes are included in the prompt; each row entry represents a single prompt.

ing the robot base to a relative SE(2) pose. Collisions are anticipated and a certain amount of local navigation around obstacles is handled automatically. Because of this, and because our environments are relatively free of obstacles, we do not require full-fledged motion planning. However, we do need to sample collision-free target positions to implement the parameterized move skills. For example, the parameter prior for `MoveTo` skills samples a distance and an angle relative to a target object; only collision-free poses should be sampled. We perform conservative collision checking in SE(2) using the known robot and object dimensions.

Pick skills. The Spot SDK provides functionality for grasping at a pixel in the hand camera image. Rotation constraints on the gripper can also be enforced. By default, our `Pick` skills select a random pixel in the target object mask returned by Detic/SAM, with no rotation constraints enforced. However, for certain objects, we implement specific grasp pixel selection logic. In principle, such logic could be learned, but doing so would require learning over images, which we do not consider in this work. Note that we *do* learn nontrivial parameter policies for grasping in the simulation experiments; see Appendix F. As an example of object-specific grasp selection, for the brush in Cleanup Playroom, we do the following: (1) Detect the largest connected component of blue pixels in the image (i.e., the center of the brush handle); (2) Choose the center pixel of that connected component for grasping; (3) Find the head of the brush with respect to the center of the pixel and set the rotation constraint so that the head is to the right of the gripper. The logic for other objects is typically less involved; see https://planning-to-practice.github.io.

3) Real-Robot Limitations and Failure Modes: Computer vision, robot skill policies, and robot hardware are all continuing to improve, and the methods proposed in this paper will continue to be applicable as they do. However, the stack we implemented has several limitations. In this section, we mention a few of those limitations and note how we worked around them for the purpose of running our experiments.

Selecting objects for Cleanup Playroom. We selected the ball and car toys used in the Cleanup Playroom environment after considering multiple constraints. The objects need to be small enough for the robot to grasp, large enough that a successful grasp can be distinguished from a fully-closed gripper, light enough for the robot to lift and sweep, heavy enough that they won't always bounce out of the bin when swept, and visually distinct enough that Detic can reliably find them (on the table, on the floor, in the bin, etc.).

Selecting (short) tables and bins. The tables and bins used in both real-robot environments are notably low to the ground. This is because the robot needs to see objects on top of the tables and inside the bins. In earlier versions of our environments, we included "look from above" actions at the end of certain skills, like placing objects on tables, by moving the robot's hand high up and looking down. But this is not possible in the Cleanup Playroom environment, where the robot is holding the brush after sweeping into the bin. Another workaround would be to place cameras or other sensors in the external environment, but we were committed to working with Spot's on-board sensors alone.

3D printing handles and adding wheels to drag objects. The blue component on the brush in Cleanup Playroom is a part that we 3D printed and added on. Without that part, the robot was not able to consistently grasp the brush from the floor and prepare to sweep without the brush slipping in-hand. Additionally, the platform and chair in the environment could not be reliably dragged without attaching specific ball-bearing wheels to their legs. These were specifically selected and attached to the objects with 3D printed parts.

Object detection failures. With Detic, we regularly encounter false positives (objects detected where they shouldn't be) and false negatives (objects not detected at all). We deal with this by prompt hacking (Table IV) and discarding detections with confidences below a threshold (0.4). Even when objects are detected correctly, there is another issue stemming from the fact that we do not assume known object models and we are not doing true pose detection. Since we use bounding box centers to define the xy position of the object in the camera frame, and since our views of the objects are constantly changing, the object's reference frame is also constantly changing. Surprisingly, this is often not an issue; the skills we use in this work are largely robust to these variations. But occasional failures can occur. For example, suppose that the robot sees only the top of an object and assigns a z position in the world frame that has the object floating above the floor. The classifier for `On(obj, floor)` would subsequently misfire, leading to initiation condition failures in skills like `Pick(obj, floor)`. In future work, we hope to integrate full-fledged pose detection into our system, but note the need for low latency and high accuracy.

E. Approach Details

In this section, we give implementation details for each of the approaches used in the paper. The following details are shared for all approaches. We alternate between task time, free time, and learning. During task time, the approaches pursue the given task goal until it is either achieved, or until H_{eval} steps have been taken. These task horizons are $H_{\text{eval}} = 27$ (the number of grid cells plus 2) for Light-Switch, 8 for Ball-Ring (Simulated), 10 for Cleanup Playroom (Simulated), 15 for Ball-Ring (Real), and 12 for Cleanup Playroom (Real). Free time then lasts until a maximum number of steps is reached: 150 for Light Switch, 100 for Ball-Ring (Simulated), 125 for Cleanup Playroom (Simulated), 20 for Ball-Ring (Real), and 50 for Cleanup Playroom (Real). For the epsilon-greedy policy in skill practicing, we use $\epsilon = 0.5$. Learning details are given in the main paper.

1) Planning-to-Practice Approaches: There are five approaches that plan to practice: EES (Ours), Fail Focus, Competence Gradient, Skill Diversity, and Task-Relevant. These approaches are identical except in their criteria for choosing a skill to practice. EES and Task-Relevant call the planner internally; EES uses the planner for the "situate" step, and Task-Relevant uses the planner to determine which skills are relevant to previously seen tasks. Re-invoking the planner on all previously seen tasks at each practice decision can be slow, so we use two optimizations. First, we use only the 10 most recently seen tasks. Second, we cache the last plan output for each task and rerun the planner only once out of every 100 calls. Note that plans cannot be cached for all time because the changing skill competences may change the maximum-likelihood plans. More sophisticated caching strategies that use the competence changes themselves are possible.

2) MAPLE-Q: MAPLE [44] requires three trained networks: (1) a *task policy* network, (2) a *parameter policy* network, and (3) a Q-network ($Q_\sigma(x, u, \theta)$). Here, x is an input state, a is a parameterized skill, and θ is a set of continuous parameters to be input to a ground skill. The task network is intended to select a skill u and the parameter policy network is intended to select a continuous parameter vector θ conditioned on the ground skill u and the state x (similar to our skill parameter policies). Together, these networks serve as an actor that outputs a ground parameterized skill that can be executed in the environment. The Q-network $Q_\sigma(x, u, \theta)$ serves as a critic that outputs Q-values given a state and skill.

Given our setting and main approach (EES), there are two significant reasons why comparing against MAPLE directly is unfair. Firstly, our approach can leverage symbolic operators (specifically their preconditions) to discern states in which particular skills are applicable, whereas the task policy in MAPLE must learn this. And secondly, our approach has access to parameter priors for each skill that can be used to produce policy parameters θ, whereas the parameter policy network in MAPLE does not.

We seek to remedy these by giving MAPLE access to our symbolic operators and parameter priors, and doing away with the task and policy networks. Specifically, given a state x, we use our operators to determine which skills can be executed from this state. For each of these skills, we sample n_{s} number of continuous parameter vectors from our parameter prior. We then pass n_{s} tuples of (x, u, θ) per applicable skill

through the Q-network and choose the maximum. Intuitively, the Q-network must not only implicitly learn good parameter policies, but also how to sequence together skills given a goal (i.e., learning how to plan).

More concretely, we train a Q-network that takes as input a vector consisting of 4 smaller vectors concatenated together: (1) a continuous vector of the features of all objects in the current state (i.e., state x), (2) a one-hot vector corresponding to the skill to be invoked from the current state, (3) a vector of the continuous parameters θ to be passed to the skill (potentially padded with zeros), (4) a one-hot vector corresponding to the current goal being solved (for environments with multiple possible goals). Given this input, the Q-network predicts a Q-value. Similar to our other approaches, we train the network with a batch size of 64 using the Adam optimizer for 10000 iterations, with early stopping after 5000 iterations of no change in the loss. We use epsilon-greedy exploration during free-time, with epsilon set to 0.5, and at test time, we sample 100 θ vectors from the parameter prior of each skill applicable in the current state and run the skill with the maximum Q-value from these.

F. Environment Details

In this section, we detail the environments used in our experiments. Note that all skills listed below have discrete object parameters (indicated by the '`?`') as well as continuous parameters θ (shown within []). Thus, the total number of skills is much larger than shown here, since these skills need to be ground with the various objects in the environment. For specific implementation details beyond what is presented here, please see https://planning-to-practice.github.io.

- *Light Switch (Simulated)*: The main challenge in this environment is for the robot to specialize its parameter prior for the `ToggleLight` skill.
  - Predicates: `RobotInCell(?robot, ?cell)`, `LightInCell(?light, ?cell)`, `LightOn(?light)`, `Adjacent(?cell1, ?cell2)`
  - Skills:
    - `MoveTo(?robot, ?cell1, ?cell2)`: Moves the robot between `?cell1` and `?cell2` provided the robot is currently in `?cell1` and `?cell2` is adjacent to it.
    - `ToggleLight(?robot, ?light, [dlight])`: Spins the light dial if the robot is currently in the same cell as the light. If the 'level' feature of the light plus the `dlight` continuous parameter value yields the `target` value for the light (which is a feature of the light), then the light will turn on. The parameter prior for `dlight` is simply a uniform distribution over $[0, 2\pi]$.
    - `JumpToLight(?robot, ?cell1, ?cell2, ?cell3, ?light)`: Tries to have the agent 'jump' directly from `?cell1` to `?cell3` given cells 1, 2, and 3 are adjacent. However, this skill is impossible and never achieves its purported effect.
  - Goal(s): Achieve `LightOn(?light)` given the robot starts in the first cell.
- *Ball-Ring (Simulated)*: Note that there is only one robot, one ball and one ring, but 5 different tables, some of which are slanted and others which are not. The main challenges in this environment are for the robot to learn that the competence of the `Place(ball, table)` skill is very low for a slanted table and cannot be improved (thus requiring it to switch its strategy for accomplishing the goal), and for it to specialize the parameter prior for the `Place(ring, table)` skill to place the ring on the high-friction part of the table so it doesn't slide down.
  - Predicates: `On(?obj, ?surface)`, `Reachable(?robot, ?obj)`, `Inside(?obj, ?container)`
  - Skills:
    - `Pick(?robot, ?obj, [x, y])`: Picks up an object if the robot is reachable to it and the robot's hand is currently empty. The pick will only succeed if the [x, y] params fall somewhere on the object's surface. The parameter prior is designed to be perfect, leading to 100% success at this action.
    - `PlaceOnTop(?robot, ?obj, ?surface, [x, y])`: If the robot is holding `?obj` and reachable to `?surface`, this skill will place the ball at the [x, y] parameters indicated. Note that it is not possible to make the ball stay on any table (it will always bounce/roll off). The parameter prior is a uniform distribution over all locations on `?surface`.
    - `PlaceInside(?robot, ?obj, ?container, [x, y])`: Similar to the above `PlaceOnTop` skill, but instead attempts to place `?obj` inside `?container`. The parameter prior is uniform over the inside surface of `?container`, which yields success 100% of the time when called from an applicable state.
    - `NavigateTo(?robot, ?obj, [x, y])`: Moves the robot to the [x, y] parameters indicated, unless they are in collision with some object. The parameter prior is a uniform circle around `?obj` such that the radius of the circle is the maximum distance at which the robot will be reachable to the object.
  - Goal(s): Achieve `On(ball, table1)` for a particular `table1` that happens to be slanted, which means the only way to succeed is to place the ring on the table, and then place the ball in the ring. The ball always starts out atop one of the tables, while the ring is on the floor. Table positions are randomized around the room.
- *Real-World Domains and Cleanup Playroom (Simulated)*: These environments all share a common set of predicates and skills. We list these first, followed by the skills and

goals specific to each particular environment.

- Predicates: `NotEqual(?obj0, ?obj1)`, `OnTop(?obj0, ?obj1)`, `Above(?obj0, ?obj1)`, `Inside(?obj0, ?obj1)`, `FitsInsideXY(?obj0, ?obj1)`, `HandEmpty(?robot)`, `Holding(?robot, ?obj0)`, `InHandView(?robot, ?obj0)`, `Reachable(?robot, ?obj0)`, `Blocking(?obj0, ?obj1)`, `IsPlaceable(?obj0)`, `IsSweeper(?obj0)`, `HasFlatTopSurface(?obj0)`, `PlatformInFrontOfSurface(?platform, ?surface)`, `SurfaceTooHigh(?surface)`, `SurfaceNotTooHigh(?surface)`, `RobotOnPlatform(?robot)`
- Skills:
  - `MoveToReachObject(?robot, ?obj0, [dist, angle])`: Moves the robot to a position that is `dist` meters away and `angle` radians from the center location of `?obj0` and is oriented such that it is facing the center of the object. The parameter prior is a uniform distribution over distances between 0.1 and 0.8 meters, and angles between 0 and 2π radians.
  - `MoveToViewObject(?robot, ?obj0, ?surface, [dist, angle])`: Same as the `MoveToReachObject`, except that it also moves the robot's hand such that it is gazing directly at `?obj0`. The parameter prior is also set to be uniform in distances between 1.1 and 1.3m (further away than `MoveToReachObject`) so the hand camera gets a clear view of the whole object.
  - `PickObjectFromTop(?robot, ?obj0, ?surface, [px, py, qx, qy, qz, qw])`: If `?obj0` is not too-high up for the robot to reach, this skill tries to pick it up by grasping at the pixel location [px, py] in the image taken from the hand camera prior to grasping and with the arm oriented according to the quaternion `[qx, qy, qz, qw]`. For most objects, the parameter prior is uniform over all pixels in the object mask, and there is no constraint places on the quaternion (allowing the skill to grasp the object in whatever orientation it chooses). For some objects, such as the chair, the parameter prior is a much narrower distribution (e.g. for the chair, `px, py` are constrained to be a point at the top of the chair back, and the quaternion is constrained to do a top down grasp).
  - `PlaceObject(?robot, ?obj0, ?surface, [dx, dy, dz])`: If the robot is currently holding an object, moves the hand to `[dx, dy, dz]` from the object's top-most center point and opens the gripper. This generally leads to a very high success rate for placing objects.
  - `PickAndDumpContainer(?robot, ?container, ?surface, ?obj, [px, py, qx, qy, qz, qw])`: Similar to the `PickObjectFromTop`, but picks up `?container` that contains `?obj0`, and then tries to dump `?obj0` so it falls onto the floor before putting `?container` back where it was picked from.

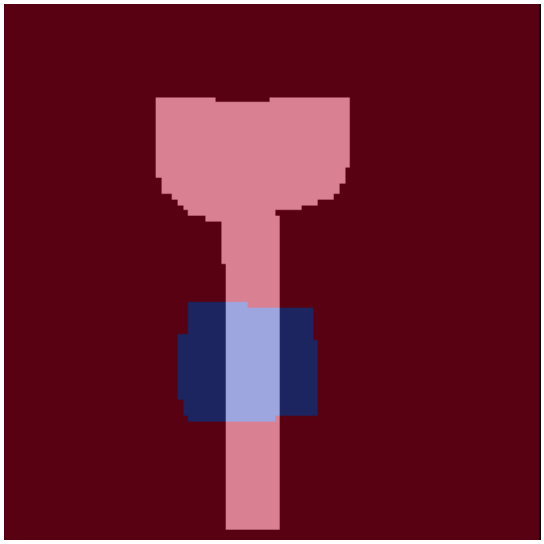

Fig. 8: **Visualization of grasping constraints for Simulated Cleanup Playroom environment**. The top-down profile of the brush object is shown in pink, while the allowed region for grasping is shown in blue. If the robot attempts to grasp the object outside the allowed region, the grasp will fail.

- *Real-World Ball-Ring*: The main challenges of this environment are identical to the simulated variant. Given that this is implemented with real perception and control, there is also non-trivial noise to be dealt with. Note that all skills used here are listed in the above entry.
  - Goal: Achieve `BallOnTable(?ball, table1)` where `table1` is the slanted table in the room. Unlike the simulated version, there is only one table, and it is slanted, with about 40% of the left side covered with a high-friction material such that the ring will stay there if placed, and the remainder left smooth so that the ring will slide down immediately.
- *Cleanup Playroom (Simulated)*: The main challenges in this environment are specializing the parameter prior to correctly learn to grasp the brush (and other objects) in their allowed regions (see Figure 8), and learning to specialize the parameter prior for the velocity to use to sweep objects successfully from various positions atop a table into a bucket.
  - Skills (in addition to those in the entry on 'Real-World Domains and Cleanup Playroom (Simulated)'):
    - `DragToUnblockObject(?robot, ?blocker, ?blocked, [dx, dy, dyaw])`: if the robot is grasping `?blocked` and `?blocked` is also being blocked by `?blocked`,

then this skill will move `?blocker` to unblock `?blocked`. This is accomplished by moving the robot by `dx, dy` in the x and y directions respectively before rotating by `dyaw`. The parameter prior is set to simply always rotate by a specific amount, since this is often enough to unblock objects in our environment.
    - `DragToBlockObject(?robot, ?blocker, ?blocked, [dx, dy, dyaw])`: Does the exact opposite of the `DragToUnblockObject` skill provided the robot starts out having already grasped `?blocker`.
    - `SweepIntoContainer(?robot, ?sweeper, ?obj, ?surface, ?container, [velocity])`: If the robot is holding `?sweeper`, is reachable to `?surface)`, `?container` is ready for sweeping, and `?obj` is atop `?surface`, then uses `?sweeper` to try to push `?target1` and `?target2` into `?container`. The sweeping motion is performed with velocity corresponding to the `velocity` parameter. The parameter prior is a uniform distribution over a range of velocities the robot arm is capable of moving at.
  - Goal(s): Achieve `ObjectInsideContainer` for two different objects that begin atop the sole table in the environment. We also experimented with only having the goal mention one object, which drastically changes the maximum likelihood task plan.
- *Real-World Cleanup Playroom*: The main difference between this environment and its simulated counterpart is that the parameter priors already provide a very high success rate for grasping the brush and other objects (since learning a grasp sampler in the real world is prohibitively sample inefficient). The main challenge is thus learning to specialize the parameter prior for sweeping real objects from the table into the bucket. Given that this is implemented with real perception and control, there is also non-trivial noise to be dealt with.
  - Skills (in addition to those from the simulated variant of this environment):
    - `DragPlatformInFrontOfSurface(?robot, ?platform, ?surface)`: If the robot is initially grasping `?platform`, drags it such that it is positioned in front of `?surface` so that the robot can stand atop it to pick up an object atop `?surface` (which is too high for the robot to ordinarily pick objects from).
  - Goal: Achieve `ObjectInsideContainer` for two different objects that begin atop the sole table in the environment.