Abstract: Lifelong-learning robots need to be able to acquire new skills and plan for new tasks over time. Prior works on planning with skills often make assumptions on the structure of skills and tasks, like subgoal skills, shared skill implementations, or learning task-specific plan skeletons, that limit their application to new and different skills and tasks. By contrast, we propose doing task planning by jointly searching in the space of skills and their parameters with skill effect models learned in simulation. Our approach is flexible about skill parameterizations and task specifications, and we use an iterative training procedure to efficiently generate relevant data to train such models. Experiments demonstrate the ability of our planner to integrate new skills in a lifelong manner, finding new task strategies with lower costs in both train and test tasks. We additionally show that our method supports planning in both state and latent spaces, and it can transfer to the real world without further fine-tuning.

1 Introduction

Lifelong-learning robots need to be able to plan with new skills and for new tasks over time [1]. For example, a home robot may initially have skills to rinse dishes and place them individually on a rack. Later, the robot might obtain a new skill of operating a dishwasher. Now the robot must plan to either wash the dishes one by one or use the dishwasher depending on the costs of each skill and the number of dishes to be cleaned. In other words, robots need to be able to obtain and use new skills over time to either adapt to new scenarios, or for new tasks that are added over time, or to improve performance on existing tasks. Otherwise, the robot designer would need to account for all potential tasks and all possible ways the robot can do them before deployment, which is impractical. To achieve such capabilities, we propose a task planning system that can efficiently incorporate new skills and plan for new tasks in a lifelong robot manipulation setting.

To create such a versatile manipulation system, we use parameterized skills that can be adapted to different scenarios by selecting suitable parameter values. We identify three properties of skills that are important to support in this context: 1) skills can have different implementations, 2) skills can have different parameters which can take discrete, continuous, or mixed values, and 3) skill parameters may or may not correspond to subgoals. Property one means the skills can be implemented in a variety of manners, e.g., hard-coded, learned without models, or optimized with models. This requires relaxing the assumptions placed on the skill structures made in previous works, like implementing all skills with the same skill-conditioning embedding space [2, 3, 4, 5]. Property two requires the task planner to not assume any fixed structure for skill parameters. Unlike previous works [6, 7], each skill can utilize a different number of parameters, and these parameters can be a mix of discrete and continuous values. Property three means that instead of chaining together skill subgoals, the planner needs to reason about the effects of the skills for different parameter values. For example, the home robot may need to predict how clean a plate is for different rinsing durations.

Planning for new tasks requires the task planner to be flexible about the structure of task specifications. Hence, a task should be described by either a goal condition function or goal distributions [8], instead of shared representations like task embeddings [9] or specific goal states [10, 6, 5, 11]. Using predefined task representations limit the type of tasks a robot can do, and using learned task
embeddings may require additional fine-tuning on new tasks. Only having access to a goal condition function also means function approximators cannot easily take in tasks as inputs, making it difficult to learn general value or policy functions for high-level planning.

To satisfy the skill and task requirements for the lifelong manipulation planning problem, we propose a task planning system that performs search-based planning with learned effects of parameterized skills. Search-based methods directly plan in the space of skill-parameter tuples. They support skills regardless of their parameter choices and implementation details, and only a general goal condition check is needed to evaluate task completion.

To efficiently use search-based planning methods in task planning, we propose to learn skill effect models (SEMs). SEMs are learned instead of hardcoded or simulated, since manually engineering models is not scalable for complex skills and simulations are too expensive to perform online during planning. Every skill has its own SEM that predicts the terminal state and costs of a skill execution given a start state and skill parameters. We interleave training SEMs with generating training data by running the planner with the learned SEMs on a set of training tasks. This efficiently collects skill execution data relevant for planning, and it supports the addition of new skills and tasks over time. The planner uses the learned SEMs to plan for existing tasks with different initial states, as well as new tasks not seen during training.

In summary, our contributions are 1) a search-based task planning algorithm with learned skill-effects models that 2) relaxes assumptions of skill and task representations in prior works; skill effect models are learned with 3) an iterative data collection scheme that efficiently collects relevant training data, and together they enable 4) planning with new skills and tasks in a lifelong manner.

2 Related Works

Subgoal Skills. Many prior works approached planning with skills with the subgoal skill assumption. The successful execution of a subgoal skill always results in the same state or a state that satisfies the same preconditions of all skills, regardless of where the skill began in its initiation set [12]. As such, the skill effects are always known, and such approaches instead focus on learning preconditions [6] of goal-conditioned policies, efficiently finding parameters that satisfy preconditions [7, 13], or learning feasible skill sequences [11]. While subgoal planning is powerful, it limits the types of skills the robot can use.

Non-Subgoal Skills. For works that plan with non-subgoal skills, many represent the skill policy as a neural network that takes as input both the state and an embedding that defines the skill. This can be viewed as planning with one parameterized skill or a class of non-parameterized skills, each defined by a different embedding. Such skills can be discovered by experience in the real world [3] and in learned models [2, 4], or learned from demonstrations [5]. Planning with these skills is typically done via Model Predictive Control (MPC), where a short sequence of continuous skill embeddings is optimized, and replanning occurs after every skill execution. While these approaches do not assume subgoal skills, they require skills to share the same implementation and space of conditioning embeddings, and MPC-style planning cannot easily support planning with multiple skills with different parameter representations [6, 4, 3, 5].

Obtaining Skill Effects. Many prior works used simulated skill outcomes during planning [19, 20, 21, 14]. This can be prohibitively expensive to perform online, depending on the complexity of simulation and the duration of each skill. To avoid simulation rollouts, works have used hardcoded
Figure 1: Overview of the proposed search-based task planning framework with learned skill effect models for lifelong robotic manipulation. New skills and new training tasks can be added incrementally. We collect skill effects data by running the planner using all skills on all training tasks in simulation. The collected data is used to train GNN skill effect models for new skills or fine-tune models for existing skills. Learned models predict both the terminal state of skill execution as well as execution cost, which enables search-based task planning that finds low-cost paths. The planner can use the learned models to plan on test tasks in the real world. This approach supports planning 1) with a set of differently parameterized skills that can grow over time and 2) for test tasks unseen during training.

analytical [22, 23] or symbolic [24, 25, 26] skill effect models. Manually engineering such models may not always be feasible, and they do not easily scale to changes in skills, dynamics, and tasks. Although symbolic models can be automatically learned [27, 12, 28, 29, 30], these approaches also make the subgoal skill assumption. By contrast, our method, which learns skill effect models in continuous state spaces without relying on symbolic constructions, can plan with both subgoal skills as well as skills that do not share this property.

The works most closely related to ours are [15] and [30]. In [15], the authors jointly train latent dynamics, latent preconditions, and parameter samplers for hardcoded skills and a model that proposes plan skeletons. Planning is done MPC-style by optimizing skill parameters with the fixed plan skeleton. Although this approach does not assume subgoal skills and supports skills with different parameters, learning task-specific plan skeletons and skill parameter samplers makes it difficult to use for new tasks without finetuning. The method in [30] learns to efficiently sample skill parameters that satisfy preconditions. Task planning is done using PDDLStream [31], which supports adding new skills and tasks. Though this approach does not use subgoal parameters, the desired skill outcomes are narrow and predefined, and the learned parameter sampler aims to achieve these predefined effects. As such, the method shares the limitations of works with the subgoal skill assumption, where the skill-level transition model is not learned but predefined as the skills’ subgoals.

Additional comparisons of related works discussed here can be found in Appendix A.

3 Task Planning with Learned Skill Effect Models

The proposed method consists of two main components - learning skill effect models (SEMs) for parameterized skills and using SEMs in search-based task planning. These two components are interleaved together - we run the planner on a set of training tasks using SEMs to generate data, which is used to further train the SEMs. New skills and training tasks can be added to the pipeline, as the planner and the SEM models do not assume particular implementations of skills and tasks. The planner can also directly use the learned SEMs to solve test tasks. See Figure 1 for an overview.

3.1 Formulation of the Skill Planning Problem

Parameterized skills. Central to our approach is the options formulation of skills. Denote a parameterized skill as $o$ with parameters $\theta \in \Theta$. Parameters are skill-specific, and they may contain subgoal information such as the target object pose for a pick-and-place skill. We assume a fully observable state $x \in X$ that contains all information necessary for task planning, cost evaluations, and skill executions. Define the low-level action $u \in U$ as the command sent to the robot by a low-level controller shared by all skills (e.g. torque).

In our formulation, a parameterized skill $o$ contains the following 5 elements: an initiation set (precondition) $I_o(x, \theta) \rightarrow \{0, 1\}$, a parameter generator that samples valid parameters from a distribution $p_o(\theta | I(x, \theta)) = 1$, a policy $\pi_o(x) \rightarrow u$, a termination condition $\beta_o(x, \theta, t) \rightarrow \{0, 1\}$, and the skill effects $f_o(x_t, \theta) \rightarrow x_{t+T}$, where $T$ is the time it took for the skill to terminate. To execute
skill \( o \) at state \( x \) with parameters \( \theta \), we first check if \((x, \theta)\) satisfies the preconditions and belong to
the initiation set \( \mathcal{I}_o \). If it does, then we run the skill’s policy \( \pi_o \) until the termination condition is satisfied. We assume that the precondition, parameter generator, policy, and termination conditions are given, and the skill effects are unknown but can be obtained via simulating the policy. To enable reasonable planning speeds, the SEMs learn to predict these skill-level transitions.

### Task planning of skills and parameters

Before specifying tasks, we first define a background, task-agnostic cost \( c(x_t, u_t) \geq 0 \) that should be minimized for all tasks. The background
cost is accumulated at each step of skill execution, so the total cost is \( c_o = \sum_{t=0}^T c(x_t, u_t) \).

A task is specified by a goal condition \( G(x) \rightarrow \{0, 1\} \) that classifies whether or not a state achieves the task. We denote a se-
quence of skills, parameters, and their incurred states as a path
\( P = (x_0, o_0, \theta_0, x_1, \ldots, x_n, o_n, \theta_n, x_{n+1}, \ldots, x_N) \), where \( N \) is the
number of skill executions, and the subscripts indicate the \( n \)th skill
in the sequence (not time). We assume the environment dynamics
and skill policies are deterministic. The task planning problem is to find a path \( P \) such that the goal condition is satisfied at the end of
the last skill, but not sooner, and the sequence of skill executions is feasible and valid. See equation 1.

### Learning SEMs for manipulation skills

We learn a separate SEM for each skill, which takes as input the current state \( x_t \) and a skill parameter \( \theta \). The SEM predicts the terminal state \( x_{t+T} \) reached
by the skill when it is executed from \( x_t \) using \( \theta \) and the total skill execution cost \( c_o \). We assume
SEMs are queried only with state and parameter tuples that satisfy the precondition. Because we
focus on the robot manipulation domain, we assume the state space \( \mathcal{X} \) can be decomposed into a list
of object-centric features that describe discrete objects or robots in the scene.

We model SEMs with Graph Neural Networks (GNNs), because their inductive bias can can effi-
cently model interactions among entities through message passing, encode order-invariance, and
support different numbers of nodes and edges during training and test [32, 33, 34, 35]. Each node in
the SEM GNN graph corresponds to an object in the scene and contains features relevant to that ob-
ject from the state \( x \). We denote these object features as \( s_k \in \mathbb{R}^S \), where \( k \) denotes the \( k \)th object in
the scene. Because a skill may directly affect multiple objects, each node also contains the skill pa-
rameters \( \theta \) as additional node features. The full node feature is the concatenation of
\( [s_k, \theta] \). There are no edge features. The network makes one node-level prediction, the change in object features \( \Delta s_k \),
and one graph-level prediction, the total skill execution cost \( c_o \). As SEMs make long-term predic-
tions about the entire skill execution, the graph is fully connected to allow all objects the possibility
of interacting with each other, not just objects that are initially nearby. The loss function to train
SEMs for a single step of skill execution prediction is \( L = \lambda_c ||c_o - \hat{c}_o||_2^2 + \sum_{k=1}^K ||\Delta s_k - \hat{\Delta s}_k||_2^2 \).

The hat notation denotes predicted quantities, and the \( \lambda \)s are positive scalars that tune the relative
weights between the loss terms. The GNN is implemented with PyTorch Geometric [36].

### Collecting diverse and relevant data for training SEMs

To learn accurate and generalizable SEMs, they must be trained on a set of skill execution data that is both diverse and relevant to task
planning. While we assume knowledge of the initial state distribution of all tasks, we do not know
the distribution of all states visited during planning. As such, we obtain these state transitions and
train the SEMs in an iterative fashion. First, given an initial set of skills, we generate single skill
execution transitions from the known initial state distribution. This data is used to train the initial
SEMs. Then, given a set of training tasks, we deploy the planner to plan for these tasks using the
learned SEMs across a set of initial states. The planner terminates when it finds a path to the goal or
reaches a fixed planning budget (reaching maximum number of nodes expanded, maximum search depth, or maximum planning time). Then we sample paths in the graph and simulate them in a high-fidelity physics simulator to collect skill execution data, which is then appended to a dataset of all skill data collected so far. Path sampling is biased toward longer paths and ones that have the newly added skills. The transitions added are filtered for duplicates, since multiple paths in a planning graph may share the same initial segments, and we do not want to bias the dataset toward transitions closer to the initial states. After a fixed amount of path data is collected, we continue training the SEMs on the updated dataset, before restarting the data collection process. In the beginning, it is expected that the planner performance will be highly suboptimal due to the inaccurate initial SEMs.

**Planning with new skills.** The above procedure supports incrementally expanding the list of skills used by the planner. Given a new skill, we first train an initial SEM by sampling from the initial state distribution, then during planning data generation the search-based planner can use the new SEM to get successors. SEMs for new and existing skills will be improved and continuously trained on this new planning data. Fine-tuning previous SEMs is needed, because the new skill might have incurred states that were previously absent from the dataset. Although this fine-tuning may not be necessary in specific cases, we leave detecting such scenarios and reducing overall training budget to future work. Learning one SEM for each skill allows for different parameter spaces (e.g. dimensions, discrete, continuous, mixed) that cannot be easily represented with a shared, common model.

**Planning with new tasks.** Because the planner does not rely on predefined plan skeletons, it can directly use SEMs to plan for new tasks. Two main factors about data collection affect the generalization capability of the SEMs when applied to unseen test tasks. The first is whether the states incurred while planning for training tasks are sufficiently diverse and relevant to cover the states incurred by planning for test tasks. The second is the planner itself — how greedy is its search and how much it explores the state space. Many planners have hyperparameters that can directly balance this exploration-exploitation trade-off.

### 3.3 Search-based Task Planning

We pose task planning as a graph search problem over a directed graph, where each node is a state \( x \), and each directed edge from \( x \) to \( x' \) is a tuple \((o, θ)\) such that \( f_o(x, θ) = x'\). Edges also contain the costs of the corresponding skill execution \( c_o \). During search, this graph is constructed implicitly. Given a node to expand, we iterate over all skills, generate up to \( B_o \) parameters per skill that satisfy the preconditions, then evaluate the skill-level dynamics on all state-parameter tuples to generate successor states. \( B_o \) decides the maximum branching factor on the graph. This number varies per skill, because some skills have a broader range of potential parameters than others, requiring more samples. The number of parameters actually sampled could be 0 if no parameters satisfy the precondition of the skill at the given state. It could also be a number between 0 and \( B_o \) if a maximum sampling budget is reached for rejection sampling with the preconditions.

To search on this graph, We apply Weighted A* (WA*), which guarantees completeness on the given graph. If the heuristic is admissible, WA* also guarantees the solution found is no worse than \( ϵc^* \), where \( c^* \) is the cost of the optimal path and \( ϵ \) determines how greedily the search follows the heuristic. We assume an admissible heuristic is given. This is in line with previous works, which often have shaped rewards or costs that guide the planner \([6, 15, 3, 5]\).

While WA* provides guarantees on a graph, we additionally need to show the constructed graph represents the underlying problem sufficiently well. Under smoothness assumptions in the dynamics and cost functions, with large but bounded \( B_o \)s the graph will contain a solution that is close to the optimal with high probability. A detailed analysis is provided in Appendix C.

The proposed search-based planning scheme enables planning with new skills and tasks. After learning the SEMs for new skills, they can be used to compute additional successors during node expansion. Planning for new tasks is done by replacing the heuristic and goal conditions, and this does not affect the graph construction procedure or the learned SEMs.

### 4 Experiments

Our main experiment studies how incrementally adding new skills to the proposed method affect planning performance on both train and test tasks. We apply our method on a blocks and bin ma-
Figure 2: The four skills in the blocks and bin manipulation domain. We evaluate our algorithm by incrementally integrating new skills over time and measure how their additions affect task planning performance.

Figure 3: Different tasks used in our experiments. The top row shows examples of initial states, the bottom shows examples of goal states. Left: blocks to bin tasks. Right: blocks to far bin tasks.

The task domain has a Franka Emika Panda 7 DoF arm, a set of colored blocks, a table, a tray, and a bin. On the table, blocks of the same size and different colors are initialized in random order on a grid with noisy pose perturbations. The tray on the table can be used as a tool to carry and sweep the blocks. Beside the table is a bin, which is divided into 2 regions, the half which is closer to the robot, and the half that is farther away. The state space contains the 3D position of each block, its color, and index. We implement the task in simulation with Nvidia Isaac Gym [41].

Skills. We experiment with four distinct skills: Pick and place (Figure 2 top-left) moves a chosen block to a target location. It has a mixed discrete and continuous parameter space — which object to pick and its placement location. Tray Slide (Figure 2 top-right) controls the robot to grasp the tray, move it over to the bin, and tilt down the tray to empty any blocks on it into the bin. Its parameter is a continuous value defining where along the length of the bin to rotate the tray. Tray Sweep (Figure 2 bottom-left) uses the tray to perform a sweeping motion along the table. Its parameter specifies where to start the sweeping motion, while the sweep motion ends at the table’s edge. Bin Tilt (Figure 2 bottom-right) controls the robot to grasp the handle at the side of the bin and tilt the bin by lifting the handle, which moves blocks in the bin from the close half to the far half.

Tasks. We use 4 different tasks, shown in Figure 3, which are variations of moving specific sets of blocks to different regions in the bin. Two tasks are used to collect SEM training data: Move All Blocks to Bin (A) and Move All Blocks to Far Bin (C), while the remaining two are used to evaluate learned SEMs: Move Red Blocks to Bin (B) and Move Red Blocks to Far Bin (D). Each task has the same background cost which is the distance the robot’s end-effector travels, plus a small penalty for placing the gripper inside the bin. The admissible heuristic used is the mean distance of each block to the closest point in their target regions. While Pick and Place can make substantial progress on all
Figure 4: Task execution costs plotted over time as new skills are learned and integrated in a lifelong manner. Blue vertical lines signify the addition of a new skill. Weighted costs are calculated by weighting the task cost with the success rate.

Figure 5: Task execution success rate for each new added skill. Each skill is being added overtime.

4.2 Lifelong Task Planning Results

To evaluate our approach for lifelong integration of new skills, we add the four skills over time using the iterative training procedure. We evaluate two scenarios, first in which the train-test task pair are respectively tasks A and B, and second with C and D. In each case, the robot starts with only Pick and Place, while Tray Slide, Tray Sweep, and Bin Tilt are added successively in that order at fixed times. We measure planning performance by execution costs, execution success, and planning times.

Figure 4 plots the execution costs over time for both scenarios. The proposed method is able to incorporate new skills over time, lowering execution costs when applicable. This is done by planning with new skills to generate new, lower-cost plans. For example, adding Tray Slide allows the planner to find plans with significantly reduced costs across all tasks, since multiple blocks can now be moved together. In other cases, adding a new skill does not drastically affect performance of a task, and the planner is able to find previously discovered plans and maintain the same execution costs. One example is when adding Bin Tilt to the blocks to anywhere in bin tasks, because the main use of the skill is to move blocks to the far side of the bin. Another is on adding Tray Sweep — it significantly reduced costs for moving all blocks to the bin but less so for moving only red blocks to the bin. This is because sweeping is only useful for the latter task when multiple red blocks line up in a column near the bin, which does not always happen due to random initial state sampling.

Figure 5 plots the probability of finding successful plans (dashed) and optimal plans (solid) with new skills. Immediately after adding a new skill, there is insufficient data to learn a robust SEM, so the planner is unlikely to find optimal plans using the new skill. Over time, as more data is collected, the SEMs improve and the probability of finding optimal plans increases. Figure 5 also shows how some tasks can only be accomplished when a new skill is incorporated. For instance, with just Pick and Place, the robot can accomplish blocks to bin tasks (A,B), but fails to plan for the blocks in far bin tasks (C,D). Adding new skills in the former case did not change the success rate of the task, which remained at 100%, although the composition of the plans found does change. For the latter case, adding Tray Slide enabled 100% success rate, while adding Tray Sweep did not affect plan compositions, but adding Bin Tilt did. These results show that our proposed method can learn skill effects and plan with SEMs in a lifelong manner, and that SEMs can plan for new tasks without additional task-specific learning. Additional qualitative results can be found in Appendix E.
Failures of the planner to find successful plans are due to inaccurate SEM predictions before SEMs are trained with sufficient data. The inaccurate next state predictions mislead the planner into planning infeasible skill sequences. Although sometimes the planner can still find paths to goal under such conditions, the plans would lead to execution failures.

**Planning with a Simulator.** To highlight the need for learning SEMs instead of simulating skill effects for task planning, we compare their planning times in Table 1. We only benchmarked the cases where the tasks are about moving all blocks and all skills are available. On average, using the learned model takes less than a second while using the simulator takes ten minutes to half an hour. Note that these results leverage the simulator’s ability to simulate many skill executions concurrently. Thus, using the simulator for more complex scenarios is prohibitively time consuming due to 1) the large branching factor when sampling many parameters for all skills and 2) a skill’s extended horizon, which is much longer than single-step low-level actions or short-horizon motion primitives. Additionally, Table 2 shows the plan times for SEMs with increasing number of skills. In all cases the planner is able to find the plans in less than half a minute.

**Training on Planner vs. Random Data.** To evaluate the benefits of using planning data for the iterative training of SEMs, we compare the test-task success rate between our approach and one that generates data by executing random skill sequences. See results in Figure 6. Training on planning data achieves higher success rates faster than training on random data, which illustrates the value of using an exploration strategy informed by planning.

**Real-world Results.** We implemented our task domain in the real world and used the learned SEMs to plan for the test task B. Three sets of planning experiments were performed, one with only **Pick and Place**, one with the addition of **Tray Slide**, and one with the addition of **Tray Sweep**. We did not implement **Bin Tilt** in the real world. Each set of experiments consists of 10 planning trials with different initial block configurations. See Table 3. These results are similar to the ones shown in the task A test curves in Figure 4. The differences are due to the small changes in real-world object locations and controller implementations. While we did not fine-tune SEMs on real-world data, this is possible and may improve real-world performance.

### Table 1: Comparing plan times (seconds) using simulator vs. SEMs.

<table>
<thead>
<tr>
<th>Task</th>
<th>Sim</th>
<th>SEMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>776.19 (46.9)</td>
<td>1.3 (0.7)</td>
</tr>
<tr>
<td>C</td>
<td>1736.8 (187)</td>
<td>0.98 (0.3)</td>
</tr>
</tbody>
</table>

### Table 2: Plan times (seconds) using SEMs for objects to bin tasks (A, B) with an increasing number of skills.

<table>
<thead>
<tr>
<th>Task +Skill</th>
<th>+Pick-Place</th>
<th>+Tray-Slide</th>
<th>+Tray Sweep</th>
<th>+Tilt Bin</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>11.3 (3.4)</td>
<td>20.2 (7.9)</td>
<td>0.6 (0.5)</td>
<td>1.3 (0.7)</td>
</tr>
<tr>
<td>B</td>
<td>7.4 (2.3)</td>
<td>14.9 (8.2)</td>
<td>18.0 (14.3)</td>
<td>22.1 (12.4)</td>
</tr>
</tbody>
</table>

### Table 3: Real-world results on Red Blocks to Bin. Costs: mean (std).

<table>
<thead>
<tr>
<th>Skill</th>
<th>Success</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pick and Place</td>
<td>1.0</td>
<td>6.68 (0.3)</td>
</tr>
<tr>
<td>+Tray Slide</td>
<td>0.9</td>
<td>3.9 (0.9)</td>
</tr>
<tr>
<td>+Tray Sweep</td>
<td>0.8</td>
<td>2.61 (0.7)</td>
</tr>
</tbody>
</table>

**Figure 6:** Success on task B with SEM trained on random vs. planner data.

8

5 **Conclusion**

We propose an approach of using search-based task planning with learned skill effect models for lifelong robotic manipulation. Our approach relaxes prior works’ assumptions on skill and task representations, which enables the planner to plan with new skills and tasks over time. Learning skill effect models improves planning speed, while the proposed iterative training scheme efficiently collects relevant data for learning. In future work, we will scale our method to larger number of skills and parameters by utilizing partial expansions to speed up planning. We will also explore probabilistic domains by using contingency planning and planning with uncertainty.
References


