2022 IEEE International Conference on Robotics and Automation (ICRA)
May 23-27, 2022. Philadelphia, PA, USA

Search-Based Task Planning with Learned Skill Effect Models for Lifelong Robotic Manipulation

Jacky Liang*[1], Mohit Sharma*[1], Alex LaGrassa[1], Shivam Vats[1], Saumya Saxena[1], Oliver Kroemer[1]

***Abstract*— Robots deployed in many real-world settings need to be able to acquire new skills and solve new tasks over time. Prior works on planning with skills often make assumptions on the structure of skills and tasks, such as subgoal skills, shared skill implementations, or task-specific plan skeletons, which limit adaptation to new skills and tasks. By contrast, we propose doing task planning by jointly searching in the space of parameterized skills using high-level skill effect models learned in simulation. We use an iterative training procedure to efficiently generate relevant data to train such models. Our approach allows flexible skill parameterizations and task specifications to facilitate lifelong learning in general-purpose domains. 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 can transfer to the real world without further fine-tuning.**

I. 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 can 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, solve new tasks, or to improve performance on existing tasks. Otherwise, the robot engineer would need to account for all potential tasks and strategies the robot can use before deployment. As such, 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, such as implementing all skills with the same skill-conditioning embedding space [2]–[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.

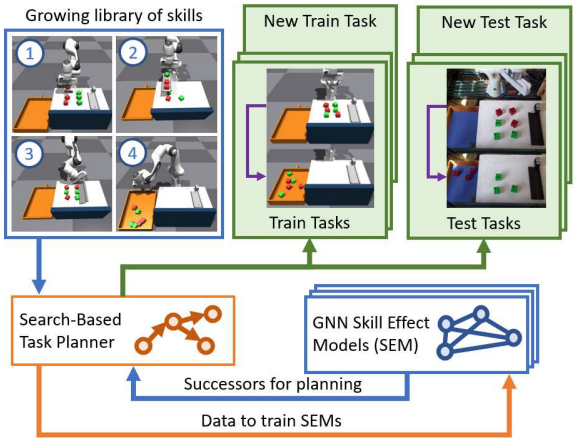

Fig. 1: Overview of the proposed search-based task planning framework with learned skill effect models (SEMs) for lifelong robotic manipulation. New skills and training tasks can be added incrementally. We collect skill effects data by running the planner using all skills on all training tasks in simulation. The collected data is used to train GNN SEMs for new skills or fine-tune models of existing skills. Learned models predict both the terminal state and cost of skill executions. The planner can use SEMs to plan low-cost paths 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.

Planning for new tasks requires the planner to be flexible about the structure of task specifications. One way to do this is by using either goal condition functions or goal distributions [8], instead of shared representations like task embeddings [9] or specific goal states [5], [6], [10], [11]. Using predefined task representations limits the type of tasks a robot can do, and using learned task embeddings may require fine-tuning on new tasks. Only having a goal condition function also makes it more difficult to represent a task as an input to a general value or policy function implemented using a function approximator.

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. A key

[1] Robotics Institute, Carnegie Mellon University, {jackyliang, mohitsharma, lagrassa, svats, saumyas, okroemer}@cmu.edu
*Equal Contribution

2022 IEEE International Conference on Robotics and Automation (ICRA) | 978-1-7281-9681-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICRA46639.2022.9811575

advantage of search-based planning methods is they can use skills regardless of parameter choices or implementation details, and only need a general goal condition check to evaluate task completion.

To efficiently use search-based planning methods for 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. Our data collection method efficiently collects skill execution data relevant for planning, and supports the addition of new skills and tasks over time. The planner uses the SEMs to plan for existing tasks with different initial states, as well as new test tasks.

Our contributions are 1) a search-based task planning framework with learned skill-effect 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. Please see supplementary materials, with additional results and experiment videos, at `https://sites.google.com/view/sem-for-lifelong-manipulation`.

II. 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 [3]–[6].

Planning with parameterized skills. To jointly plan sequences of different skills and parameters, works have proposed a two-stage approach, where the planner first chooses the skills, then optimizes skill parameters [13]–[15]. Unlike directly searching with skills and parameters, it is difficult for two-stage approaches to give guarantees on solution quality. Some also require hardcoded or learned plan skeletons [13], [15], which limits the planner's applicability to new tasks.

Instead of planning, an alternative approach is to learn to solve Markov Decision Processes (MDPs) with parameterized skills [16]–[18]. However, learning value or policy functions typically requires a fixed representation for function approximators, so these methods cannot easily adapt to new skills and skills with parameters with different dimensions or modalities (e.g. mixed continuous and discrete). Doing so for search-based planning can be done by directly appending new skills when expanding a node for successors.

Obtaining skill effects. Many prior works used simulated skill outcomes during planning [14], [19]–[21]. 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 analytical [22], [23] or symbolic [24]–[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 [12], [27]–[30], these approaches also make the subgoal skill assumption. By contrast, our method, which learns skill effect models in continuous states without relying on symbols, 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 subgoal skills, where the skill-level transition model is not learned but predefined as the subgoals.

III. 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 iteratively to further train the SEMs. New skills and training tasks can be added to the pipeline because the planner and the SEMs do not assume particular implementations of skills and tasks. The planner can also directly apply the learned SEMs to solve test tasks. See overview in Figure 1.

A. Skill Planning Problem Formulation

Parameterized skills. Central to our approach is the options formulation of skills [12], [32]. Denote a parameterized skill as o with parameters $\theta \in \Theta$. Parameters are skill-specific and may contain subgoal information such as the target object pose for a pick-and-place skill. We assume a fully observable state $x \in \mathcal{X}$ that contains all information necessary for task planning, cost evaluations, and skill executions. We define the low-level action $u \in \mathcal{U}$ 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) $\mathcal{I}_o(x, \theta) \to \{0, 1\}$, a parameter generator that samples valid parameters from a distribution $p_o(\theta|\mathcal{I}(x, \theta_i) = 1)$, a policy $\pi_o(x) \to u$, a termination condition $\beta_o(x, \theta, t) \to \{0, 1\}$, and the skill effects $f_o(x_t, \theta) \to x_{t+T}$, where T is the time it took for the skill to terminate. To execute skill o at state x with parameters θ, we first check if (x, θ) satisfies the preconditions $\mathcal{I}_o$. If it does, then we run the skill's policy π_o until the termination condition is satisfied. We assume that the preconditions, parameter generator, policy, and termination conditions are given, and the skill effects are unknown but can be obtained by simulating the policy. To enable reasonable planning speeds, the SEMs learn to predict these skill-level transitions.

To justify the assumption of given skill preconditions, we note that our preconditions are broader than ones in prior works and consequently can be easily manually defined. Preconditions in many prior works, especially ones that use subgoal skills, are only satisfied when a specific outcome is reached, so they may require learning sophisticated functions to classify which (state, parameter) tuple lead to the intended outcome [15], [30]. By contrast, because we allow non-subgoal skills, our preconditions are satisfied if skill execution leads to any non-trivial and potentially desirable outcome. For example, for a table sweeping skill, the preconditions are satisfied as long as the robot sweeps something, instead of requiring sweeping specific objects into specific target regions. Due to the broad and simple nature of our more flexible preconditions, we argue it is reasonable to assume they are given.

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. This cost is accumulated at each step of skill o execution, so the total skill cost is $c_o = \sum_{t=0}^{T} c(x_t, u_t)$. A task is specified by a goal condition $\mathcal{G}(x) \to \{0, 1\}$ that classifies whether or not a state achieves the task. We denote a sequence 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 nth 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.

$$\begin{aligned} \min_{P} \quad & \sum_{n=0}^{N-1} c_{o_n} \\ \text{s.t.} \quad & \mathcal{G}(x_N) = 1 \\ \forall n \in [0, N-1], & \mathcal{G}(x_n) = 0 \\ & \mathcal{I}_{o_n}(x_n, \theta_n) = 1, f_{o_n}(x_n, \theta_n) = x_{n+1} \end{aligned} \tag{1}$$

Note that θ, $\mathcal{I}_o$, and f_o are all skill-specific, so with M types of skills, there are M different parameter spaces, preconditions, and skill effects.

B. Learning Skill Effect Models (SEMs)

Defining 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 θ. The SEM predicts the terminal state x_{t+T} reached by the skill when it is executed from x_t using θ 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 represent SEMs using Graph Neural Networks (GNNs), because their inductive bias can efficiently model interactions among entities through message passing, encode order-invariance, and support different numbers of nodes and edges during training and testing [33]–[36]. Each node in the SEM GNN corresponds to an object in the scene and contains features relevant to that object from the state x. We denote these object features as $s_k \in \mathbb{R}^S$, where k denotes the kth object in the scene. Because a skill may directly affect multiple objects, each node also contains the skill parameters θ 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 Δs_k, and one graph-level prediction, the total skill execution cost c_o. As SEMs make long-term predictions about the entire skill execution, the graph is fully connected to allow all objects to interact 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 $\mathcal{L} = \lambda_c \|c_o - \hat{c_o}\|_2^2 + \frac{\lambda_s}{K} \sum_{k=1}^{K} \|\Delta s_k - \hat{\Delta s_k}\|_2^2$. The hat notation denotes predicted quantities, and the λs are positive scalars that tune the relative weights between the loss terms. The GNN is implemented with PyTorch Geometric [37].

SEMs enable efficient planning of diverse parameterized skills, as well as two additional benefits. First, because the model is on the skill-level, not action-level, it only needs one evaluation to predict the effects of a skill execution, which reduces planning time as well as covariate shift by reducing

the number of sequential predictions [38]–[41]. Second, a long-horizon skill-level model can leverage a skill's ability to act as a funnel in state space during execution, which simplifies the learning problem.

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 and execution. As we cannot manually specify this incurred state distribution, we obtain it and train the SEMs in an iterative fashion that interleaves SEM training with data generation by planning and execution, as seen in Figure 1. 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 use 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 to collect skill execution data, which is added 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 which would bias the dataset towards 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. While we use simulation data due to benefits in speed, this is not a requirement and SEMs can be trained with real-world data.

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.

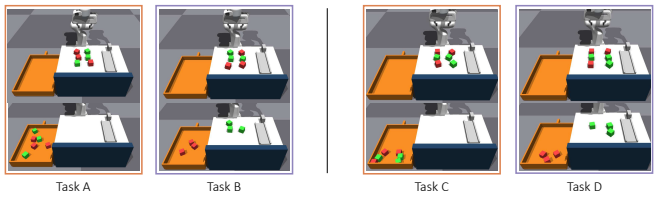

Fig. 2: Different tasks used in our experiments. The top row shows examples of initial states, the bottom shows examples of goal states. *Left:* blocks to bin tasks (tasks *(A,B)*). *Right:* blocks to far bin tasks (tasks *(C,D)*).

C. 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 $f_o(x, \theta) = x'$. Edges also contain the costs of skill executions 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.

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 that have shaped rewards or costs that guide the planner [3], [5], [6], [15].

The proposed method enables planning with new skills and to solve new tasks in continuous states. Planning for new tasks is done by replacing the heuristic and goal conditions, which does not affect the graph construction procedure or the SEMs. Searching in continuous states is more flexible than searching in symbolic states, and it is not necessarily slower. Flexibility comes from the ability to integrate new skills and tasks without needing to create new symbols. Planning speed depends on the size of the action space (branching factor) and the state space. Using symbolic instead of continuous states does not reduce the branching factor, and partitioning continuous states into symbolic states without subgoal skills yield little benefits [12].

IV. Experiments

Our main experiment analyzes the effect of incrementally adding new skills to the proposed method on planning performance of both train and test tasks. We apply our method to a block manipulation domain (Figure 2) because it can be reliably simulated, contains a diverse set of skills, and

the skills have broader applications in desktop manipulation and tool use. In addition, we show our approach compares favorably against planning with simulation and the benefits of using planning data to train SEMs. Lastly, we show the generalizability of our method by deploying it in a real-world setup. More experiment details are in Appendix-II.

A. Task Domain

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 two 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, color, and index. We implement the task domain in Nvidia Isaac Gym [42], a GPU-accelerated robotics simulator [43] that enables fast parallel data collection.

Skills. We experiment with four skills: *Pick and Place* (Figure 1 skill 1) 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 1 skill 2) grasps the tray, moves it to the bin, and tilts it down, emptying 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 1 skill 3) uses the tray to perform a sweeping motion along the table. Its parameter specifies where to start the sweeping motion, and the sweep motion ends at the table's edge. *Bin Tilt* (Figure 1 skill 4) grasps the handle at the side of the bin and tilts the bin by lifting the handle, which moves blocks in the bin from the close half to the far half. Skills are implemented by following open-loop trajectories defined by the skill parameters. We did not learn more complex skills as our work focuses on task planning and not skill learning.

Tasks. We evaluate on four different tasks (Figure 2) that 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 uses the same background cost function, 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 tasks, it alone is not sufficient because kinematic constraints inhibit the robot from directly placing blocks on the far side of the bin, so *Bin Tilt* or *Tray Slide* is needed. Additionally, using other skills can achieve lower costs; *Tray Sweep* can quickly move multiple blocks into the bin, but this may move blocks that need to stay on the table. The sequence of skills may change depending on the initial placement of the blocks, and the path needs to be low-cost.

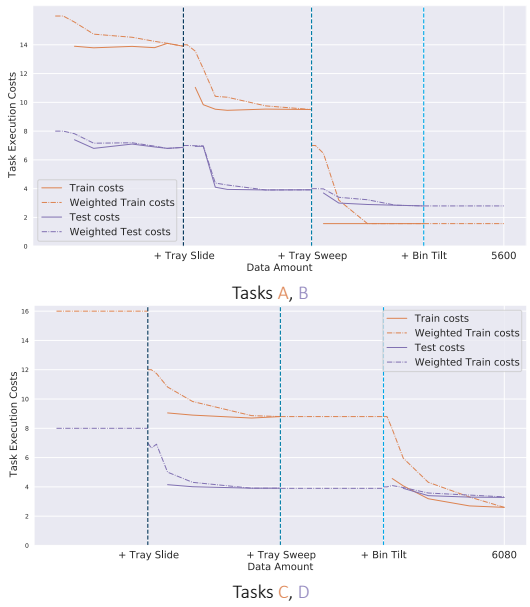

Fig. 3: Task execution costs plotted over time as new skills are learned and integrated in a lifelong manner. Blue vertical lines signify the addition of a new skill. Weighted costs are calculated by weighting the task cost with the success rate.

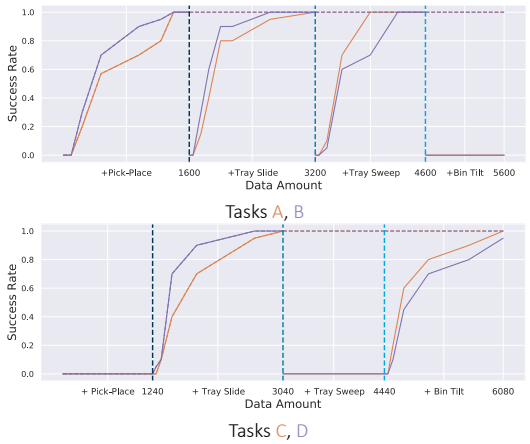

Fig. 4: Task execution success rate for each new added skill. Each skill is being added over time. Orange are train tasks; purple are test tasks. Solid lines are planning with new skills; dashed are with any skills.

B. 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 pre-determined intervals. We measure planning performance using execution costs, execution success, and planning time. For each goal, the robot plans only once from the initial state, which terminates when it succeeds or times out.

Figure 3 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 by planning with new skills. For example, adding *Tray Slide* allows the planner to find plans with significantly reduced

Task	Sim	SEMs (Ours)
A	776.19 (46.9)	1.3 (0.7)
C	1736.8 (187.)	0.98 (0.3)

TABLE I: Comparing plan times in seconds using simulator vs. SEMs. Parenthesis indicate standard deviations.

Task	Pick-Place	+Tray-Slide	+Tray Sweep	+Tilt Bin
A	11.3 (3.4)	20.2 (7.9)	0.6 (0.5)	1.3 (0.7)
B	7.4 (2.3)	14.9 (8.2)	18.0 (14.3)	22.1 (12.4)

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

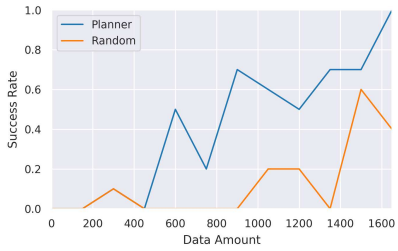

Fig. 5: Success on task *B* with SEM trained on random vs. planner data.

costs across all tasks, since multiple blocks can now be moved together. In other cases, adding a new skill does not affect task performance. One example is adding *Bin Tilt* to the blocks to anywhere in bin tasks *(A,B)*, 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 *(A,C)*, but less so for moving only red blocks to the bin *(B,D)*. This is because sweeping is only useful for the latter task when multiple red blocks line up in a column near the bin, which rarely occurs in the randomly initialized states.

Figure 4 plots the success rate 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. Or, if it does find a plan, the plan often leads to execution failures. As more data is collected, SEM accuracy improves and the probability of finding optimal plans increases. Figure 4 also shows how some tasks can only be completed after 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 for *(A,B)* did not change the success rate of the task, which remained at 100%, although the composition of the plans found does change. For *(C,D)*, 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. Qualitative results can be found in Appendix V.

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 I. We only benchmarked 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 and 2) a skill's extended horizon, which is much longer than single-step low-level actions or short-horizon motion primitives. Additionally, Table II shows the plan times for SEMs with increasing number of skills. In all cases our planner find plans in less than half a minute.

	Success	Cost
Pick and Place	1.0	6.68 (0.3)
+Tray Slide	0.9	3.9 (0.9)
+Tray Sweep	0.8	2.61 (0.7)

TABLE III: Real-world results on *Red Blocks to Bin*. Costs: mean (std).

Training on Planning Data 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 5. Training on planning data achieves higher success rates using fewer samples than training on random data does, illustrating the benefit of guiding data collection using a planner.

Real-world Results. We built our task domain in the real world (test tasks in Figure 1) 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*. Each set of experiments in Table III consists of 10 planning trials with different initial block configurations. These results are similar to the ones shown in the task *A* test curves in Figure 3. 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, doing so may improve real-world performance.

V. Conclusion

We propose using search-based task planning with learned skill effect models (SEMs) for lifelong robotic manipulation. Our approach relaxes prior works' assumptions on skill and task representations, enabling planning with more diverse skills and solving new tasks over time. Using SEMs improves planning speed, while the proposed iterative training scheme efficiently collects relevant data for training.

In future work, we will scale our method to larger number of skills and parameters by using partial expansions and learned parameter samplers. We will also explore estimating model uncertainty, using that to both steer planning away from uncertain regions and also fine-tune existing SEMs only on data about which the models are sufficiently uncertain.

VI. Acknowledgment

The authors thank Kevin Zhang for assistance on real-world experiments. This work is supported by NSF Grants No. DGE 1745016, IIS-1956163, and CMMI-1925130, the ONR Grant No. N00014-18-1-2775, ARL grant W911NF-18-2-0218 as part of the A2I2 program, and Nvidia NVAIL.

REFERENCES

[1] S. Thrun and T. M. Mitchell, "Lifelong robot learning," *Robotics and autonomous systems*, vol. 15, no. 1-2, pp. 25–46, 1995.

[2] A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman, "Dynamics-aware unsupervised discovery of skills," *International Conference on Learning Representations (ICLR)*, 2019.

[3] K. Lu, A. Grover, P. Abbeel, and I. Mordatch, "Reset-free lifelong learning with skill-space planning," *International Conference on Learning Representations (ICLR)*, 2021.

[4] K. Xie, H. Bharadhwaj, D. Hafner, A. Garg, and F. Shkurti, "Skill transfer via partially amortized hierarchical planning," *International Conference on Learning Representations (ICLR)*, 2021.

[5] T. Li, R. Calandra, D. Pathak, Y. Tian, F. Meier, and A. Rai, "Planning in learned latent action spaces for generalizable legged locomotion," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2682–2689, 2021.

[6] S. Nasiriany, V. H. Pong, S. Lin, and S. Levine, "Planning with goal-conditioned policies," *Advances in Neural Information Processing Systems*, 2019.

[7] A. Mandlekar, F. Ramos, B. Boots, S. Savarese, L. Fei-Fei, A. Garg, and D. Fox, "Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data," in *2020 IEEE International Conference on Robotics and Automation*. IEEE, 2020, pp. 4414–4420.

[8] A. Conkey and T. Hermans, "Planning under uncertainty to goal distributions," *arXiv preprint arXiv:2011.04782*, 2020.

[9] S. Nasiriany, V. H. Pong, A. Nair, A. Khazatsky, G. Berseth, and S. Levine, "Disco rl: Distribution-conditioned reinforcement learning for general-purpose policies," *International Conference on Robotics and Automation*, 2021.

[10] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal planning networks: Learning generalizable representations for visuomotor control," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4732–4741.

[11] B. Ichter, P. Sermanet, and C. Lynch, "Broadly-exploring, local-policy trees for long-horizon task planning," *arXiv preprint arXiv:2010.06491*, 2020.

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

[13] A. Simeonov, Y. Du, B. Kim, F. R. Hogan, J. Tenenbaum, P. Agrawal, and A. Rodriguez, "A long horizon planning framework for manipulating rigid pointcloud objects," *Conference on Robot Learning (CoRL)*, 2020.

[14] Z. Pan and K. Hauser, "Decision making in joint push-grasp action space for large-scale object sorting," *International Conference on Robotics and Automation*, 2020.

[15] D. Xu, A. Mandlekar, R. Martín-Martín, Y. Zhu, S. Savarese, and L. Fei-Fei, "Deep affordance foresight: Planning through what can be done in the future," *arXiv preprint arXiv:2011.08424*, 2020.

[16] W. Masson, P. Ranchod, and G. Konidaris, "Reinforcement learning with parameterized actions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[17] M. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," *International Conference on Learning Representations (ICLR)*, 2015.

[18] J. Xiong, Q. Wang, Z. Yang, P. Sun, L. Han, Y. Zheng, H. Fu, T. Zhang, J. Liu, and H. Liu, "Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space," *arXiv preprint arXiv:1810.06394*, 2018.

[19] S.-K. Kim and M. Likhachev, "Parts assembly planning under uncertainty with simulation-aided physical reasoning," in *2017 IEEE International Conference on Robotics and Automation*. IEEE, 2017, pp. 4074–4081.

[20] H. Song, J. A. Haustein, W. Yuan, K. Hang, M. Y. Wang, D. Kragic, and J. A. Stork, "Multi-object rearrangement with monte carlo tree search: A case study on planar nonprehensile sorting," *International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[21] S.-K. Kim, O. Salzman, and M. Likhachev, "Pomhdp: Search-based belief space planning using multiple heuristics," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 734–744.

[22] A. Bagaria, J. Crowley, J. W. N. Lim, and G. Konidaris, "Skill discovery for exploration and planning using deep skill graphs," 2020.

[23] J. Butzke, K. Sapkota, K. Prasad, B. MacAllister, and M. Likhachev, "State lattice with controllers: Augmenting lattice-based path planning with controller-based motion primitives," *IEEE International Conference on Intelligent Robots and Systems*, 2014.

[24] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 1470–1477.

[25] ——, "Integrated task and motion planning in belief space," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

[26] M. Eppe, P. D. Nguyen, and S. Wermter, "From semantics to execution: Integrating action planning with reinforcement learning for robotic causal problem-solving," *Frontiers in Robotics and AI*, vol. 6, p. 123, 2019.

[27] E. Ugur and J. Piater, "Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning," in *2015 IEEE International Conference on Robotics and Automation*. IEEE, 2015, pp. 2627–2633.

[28] B. Ames, A. Thackston, and G. Konidaris, "Learning symbolic representations for planning with parameterized skills," *2018 IEEE International Conference on Intelligent Robots and Systems*, 2018.

[29] A. Suárez-Hernández, T. Gaugry, J. Segovia-Aguas, A. Bernardin, C. Torras, M. Marchal, and G. Alenyà, "Leveraging multiple environments for learning and decision making: a dismantling use case," *IEEE International Conference on Intelligent Robots and Systems*, 2020.

[30] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Learning compositional models of robot skills for task and motion planning," *The International Journal of Robotics Research*, vol. 40, no. 6-7, pp. 866–894, 2021.

[31] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 440–448.

[32] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[33] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," *Advances in Neural Information Processing Systems*, 2016.

[34] M. Janner, S. Levine, W. T. Freeman, J. B. Tenenbaum, C. Finn, and J. Wu, "Reasoning about physical interactions with object-oriented prediction and planning," *International Conference on Learning Representations (ICLR)*, 2019.

[35] R. Kartmann, F. Paus, M. Grotz, and T. Asfour, "Extraction of physically plausible support relations to predict and validate manipulation action effects," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3991–3998, 2018.

[36] A. E. Tekden, A. Erdem, E. Erdem, T. Asfour, and E. Ugur, "Object and relation centric representations for push effect prediction," *arXiv preprint arXiv:2102.02100*, 2021.

[37] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[38] M. Y. Seker, A. E. Tekden, and E. Ugur, "Deep effect trajectory prediction in robot manipulation," *Robotics and Autonomous Systems*, vol. 119, pp. 173–184, 2019.

[39] P. Naderian, G. Loaiza-Ganem, H. J. Braviner, A. L. Caterini, J. C. Cresswell, T. Li, and A. Garg, "C-learning: Horizon-aware cumulative accessibility estimation," *International Conference on Learning Representations (ICLR)*, 2021.

[40] M. Janner, I. Mordatch, and S. Levine, "gamma-models: Generative temporal difference learning for infinite-horizon prediction," *Advances in Neural Information Processing Systems*, 2020.

[41] N. O. Lambert, A. Wilcox, H. Zhang, K. S. Pister, and R. Calandra, "Learning accurate long-term dynamics for model-based reinforcement learning," *arXiv preprint arXiv:2012.09156*, 2020.

[42] Nvidia. Isaac gym. [Online]. Available: developer.nvidia.com/isaac-gym

[43] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox, "Gpu-accelerated robotic simulation for distributed reinforcement learning," *Conference on Robot Learning (CoRL)*, 2018.

[44] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[45] A. Hinrichs, D. Krieg, R. J. Kunsch, and D. Rudolf, "Expected dispersion of uniformly distributed points," *Journal of Complexity*, vol. 61, p. 101483, 2020.