SPIN: distilling Skill-RRT for long-horizon prehensile and non-prehensile manipulation

Haewon Jung[*], **Donguk Lee**[*], **Haecheol Park, JunHyeop Kim, Beomjoon Kim**
Korea Advanced Institute of Science and Technology (KAIST)
{zora07, du.lee, hcp5004, amos0204, beomjoon.kim}@kaist.ac.kr

Abstract: Current robots struggle with long-horizon manipulation tasks requiring sequences of prehensile and non-prehensile skills, contact-rich interactions, and long-term reasoning. We present SPIN (**S**kill **P**lanning to **IN**ference), a framework that distills a computationally intensive planning algorithm into a policy via imitation learning. We propose Skill-RRT, an extension of RRT that incorporates skill applicability checks and intermediate object pose sampling for solving such long-horizon problems. To chain independently trained skills, we introduce *connectors*, goal-conditioned policies trained to minimize object disturbance during transitions. High-quality demonstrations are generated with Skill-RRT and distilled through noise-based replay in order to reduce online computation time. The resulting policy, trained entirely in simulation, transfers zero-shot to the real world and achieves over 80% success across three challenging long-horizon manipulation tasks and outperforms state-of-the-art hierarchical RL and planning methods. Videos are available on our project page.

Keywords: Robot Skill Chaining, Imitation Learning, Planning

1 Introduction

Humans have a remarkable capability to use sequences of prehensile (P) and non-prehensile (NP) manipulation in everyday life. For example, to take a book from a crowded shelf and place it in a tight space where our entire hand would not fit, we topple the book to enable grasping, pick-and-place it to the edge of the target space, and then push it into place using just our fingers. Our ultimate goal is to develop a policy that efficiently solves such ***p**rehensile–**n**on-**p**rehensile (PNP)* tasks that require moving an object from a random initial state to a random target pose using a sequence of P and NP manipulation skills, as illustrated in Figure 1. Such problems are extremely complex: they demand reasoning not just about the feasibility of each skill but also how to *chain* them, such as toppling a book to enable a grasp.

One way to address this problem is learning-based approaches, such as reinforcement learning (RL) and imitation learning (IL), which have shown promise in recent years [1, 2, 3, 4, 5, 6, 7, 8, 9]. However, PNP problems require reasoning over a large number of actions (e.g., more than 500 joint torques) and RL struggles with such long-horizon tasks [10]. These issues are especially pronounced in our tasks, where the object must temporally move away from the goal before ultimately reaching it, such as sliding a card to the edge of a table to enable grasping (Figure 1 (b), top row). In these problems, reward design is especially difficult. IL, in contrast, sidesteps the exploration problem through dense supervision from human demonstrations. However, it scales poorly to a wide variety of initial states and goals, because collecting large-scale, long-horizon demonstrations is expensive [6, 11, 12].

Rather than operating in a low-level action space such as joint torques, we propose to use temporally-abstracted skills to solve PNP problems. Specifically, we leverage the recent P and NP skill learning algorithms to acquire the skills [3, 13, 14, 15, 16, 17, 18, 19], and chain them together to solve PNP problems. We assume each skill has a parameterized goal-conditioned policy that takes as an input

[*]Equal contribution.

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

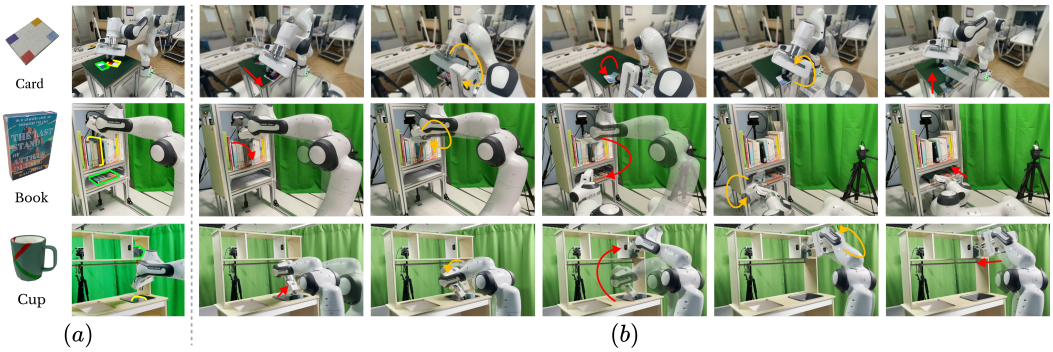

Figure 1: Overview of our tasks. (a) Objects and problems. A problem is defined by initial and goal object poses marked with yellow and green. (b) Solutions for these problems. (first row) The robot must flip a thin card, initially in an ungraspable pose, by first moving it to the end of the table to make a space for grasping, flipping it, and finally sliding it to the target pose. (Second row) The robot must put the book in the lower shelf where the robot gripper cannot fit by first toppling the book to enable grasping, picking and placing it to the end of the lower shelf, and then pushing it inside. (Third row) The robot must upright the cup in a sink, grab it, place it on the cupboard, and ensure the handle is inside the desired region by re-orienting it. The red arrow indicates the movement of the object, and the orange arrow represents the movement of the robot.

a desired object pose and state, and outputs a sequence of joint torques to achieve that pose. One way to solve PNP problems is to formulate the problem as a parameterized-action Markov decision process (PAMDPs) [20, 21, 22, 23, 24] where each parameterized action is a skill that takes in parameters, and use RL algorithms. While this shortens the planning horizon, the exploration is still challenging because PNP problems usually involve a narrow space of subgoals the policy must go through in order to achieve the goal, commonly known as *narrow passages* [25]. Alternatively, we can use planning algorithms [26], such as RRT [27], which is a time-tested method for finding a very long sequence of robot configurations in a high dimensional space. However, while RRT can solve problems involving narrow passages, it typically requires a long online computation time, making it impractical in the real world.

Therefore, we propose a two-stage framework for PNP problems, called `SPIN` (**S**kill **P**lanning to **IN**ference), that consists of (1) a *data-generation stage* that generates a large number of solutions to PNP problems using a planner inside a simulator, and (2) a *distillation stage* for learning a policy using IL to reduce online computation time and enable zero-shot real-world deployment. To do this, we propose a novel planner, `Skill-RRT`, that extends RRT [27] into the space of skills and object goals. One challenge in designing `Skill-RRT` is that typical P and NP learning algorithms are not inherently designed to learn skills in a way they would chain and there is a large gap between the resulting state of one skill and the applicable states of the next state [28]. For example, in the card domain in Figure 1 (Top row), an NP sliding skill ends with the robot's gripper closed and on top of the card, as shown in Figure 1 (b) first column. However, a P skill requires the robot to move to a pre-grasp configuration at the side of the card with the gripper opened as shown in Figure 1 (b) second column. One naïve way to solve this state gap problem is to use a collision-free motion planner to move the gripper. However, in these problems, even a small control error can disturb the object and cause it to fall, and even a slight object movement can invalidate the planned path. Instead, we need a method that intelligently breaks and makes contacts.

To address this, we introduce *connectors*—goal-conditioned policies that, given the current state and goal end-effector pose, break contact and move to the goal pose with minimal disturbance to the object's pose. For each skill, we associate a connector that brings the robot to one of the skill's applicable states, similar to Mason's concept of "funnels" [29]. We use RL to train the connectors, and to generate relevant problems for training, we propose `Lazy Skill-RRT`, which tentatively assumes the existence of connectors and teleports the robot to the next skill's applicable state. We apply `Lazy Skill-RRT` to solve PNP problems while logging the states where the robot was teleported – indicating the need for a connector. We then train the connector on these logged problems, allowing us to focus training only on states relevant to the current problem. Equipped with connectors, we use `Skill-RRT` to generate complete solutions to PNP problems.

To distill Skill-RRT to a policy, we use Diffusion Policy [5] in order to cope with multi-modality in solutions. One challenge in distilling a planner to a policy, as already observed in the previous paper [30], is that not all planning solutions are useful, as low-quality trajectories may degrade policy performance by leading the robot to a high-risk state. To mitigate this, we propose filtering data by replaying the Skill-RRT plans with noise in simulation, and discard those whose success rate is lower than a threshold. We evaluate our distilled policy both in simulation and the real world, and show that it outperforms pure planner Skill-RRT, a state-of-the-art PAMDPs-based RL algorithm MAPLE [24], state-of-the-art goal-conditioned HRL algorithm HLPS [31], and PPO [32] in simulation, and achieves 80% success rates in the real world in domains shown in Figure 1. Our supplementary video[†] further illustrates the effectiveness of the proposed method in challenging environments.

2 Related Works

Learning prehensile and non-prehensile skills There are two common approaches to learning P skills. The first approach is integrating grasp pose prediction and motion planning [13, 15, 16] where the grasp predictor outputs a grasp pose and low-level robot motion planning is used. The second approach is RL-based end-to-end policy training [1, 14, 33]. In this work, we train prehensile skills with predefined grasps using RL due to its robustness. For NP skills, recent studies use learning-based approaches where several works focus on planar pushing [34, 35, 36, 37], some on training a policy to reposition the object to enable grasping [17] and some on moving the object to a target 6D pose [19]. However, most of these approaches have a restricted motion repertoire, relying primarily on planar pushing or hand-crafted primitives. In this work, we use the method by Kim et al. [18], an RL-based approach based on a general action space (6D end-effector pose) that automatically discovers various skills such as toppling, sliding, pushing, pulling, etc.

Hierarchical reinforcement learning A widely studied approach for HRL is the *options framework* [38, 39, 40], where the agent learns options, each of which consists of a policy that operates over multiple steps, termination probability, and initiation set to achieve temporal abstraction. In practice, however, option learning often collapses into discovering primitive, one-step actions, failing to realize the intended temporal abstraction [41, 42, 43]. An alternative direction is *goal-conditioned HRL* [31, 44, 45], where a high-level policy proposes subgoals and a low-level policy achieves them using primitive actions. While these methods have shown some promising results, they often suffer from non-stationarity during training [31], as the high-level policy must continuously adapt to the evolving behavior of the low-level policy. These methods also face structural difficulties: the subgoals proposed by the high-level policy may be unachievable by the low-level policy. Unlike these works, we pre-train P and NP skills using existing approaches [18, 46, 47] to achieve temporal abstraction, rather than simultaneously learning the skills and learn to do the task.

Reinforcement learning with PAMDPs PAMDPs [20] model decision-making with a discrete set of parameterized high-level actions, each associated with its own continuous parameters (e.g., a target pose). This formulation has become increasingly viable due to recent advances in skill learning, which provide reusable low-level behaviors that generalize across a range of tasks [20, 21, 22, 23, 24, 48, 49, 50]. However, most of these methods assume that there exist states where the terminal state of one skill can directly serve as an applicable state for the next skill. To address state gaps between skills, RAPS [23], MAPLE [24], and TRAPs [48] introduce a low-level primitive that directly controls single-timestep end-effector motions in addition to a library of skills. However, as noted in [24], training such fine-grained motion policies is challenging especially when we have a large state gap and requires a long sequence of low-level actions. In contrast, we use Lazy Skill-RRT to explicit gather problems in which state gap is an issue, and train connectors with dense reward function to address this.

Task and motion planning An alternative to learning-based skill sequencing is task and motion planning (TAMP). TAMP integrates discrete task planning with continuous motion planning to solve

[†]Video link: https://www.youtube.com/watch?v=17u5FhO8eso

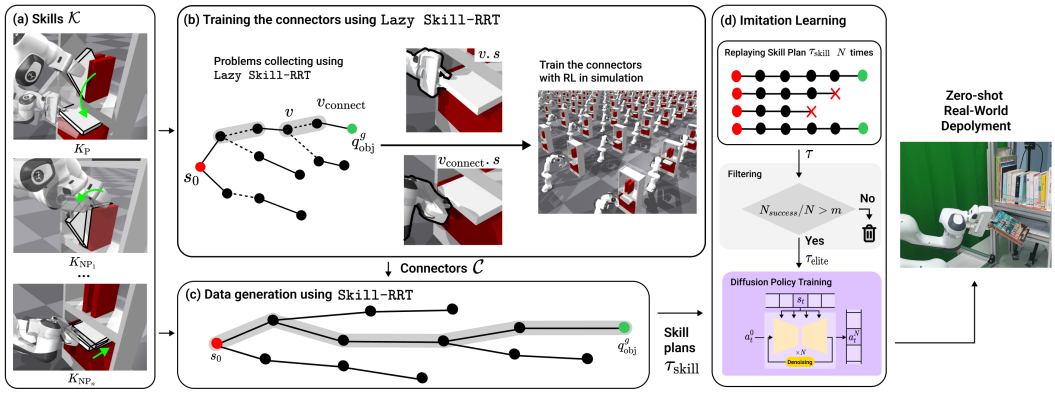

Figure 2: Overview of SPIN: (a) Examples of pre-trained PNP skills in the bookshelf domain (Figure 1, row 2). Includes pick-and-place, toppling, and pushing. (b) We first use Lazy Skill-RRT to collect problems for training connectors. Left of (b) shows the RRT tree, where the initial and goal states are marked with red and green respectively. Each edge is defined by a skill execution, and the dotted edge and its starting and end vertices, denoted v and v_{connect} respectively, defines a state gap a connector needs to fill in. The middle of (b) shows the state gap. In $v.s$, the robot has just finished a prehensile skill, with the object still in between the gripper. In $v_{\text{connect}}.s$, the robot is about to begin a pushing skill to push the book into the shelf. A connector has to fill in this state gap. We collect a set of such problems, and use RL to train connectors. (c) Skill-RRT is run with the trained connectors $\mathcal{C}$ and the skill library $\mathcal{K}$ to generate a skill plan τ_{skill}. A skill plan consists of a sequence of skills, connectors, and their associated desired object poses, or desired robot configuration. (d) We use IL to distill skill plans into a single policy. To filter data, we replay each skill plan N times, and those with a replay success rate below a predefined threshold m are filtered out. The remaining high-quality trajectories are used to train a diffusion policy, which is zero-shot deployed in the real world.

long-horizon problems by leveraging predefined skills with known preconditions and effects [51, 52, 53, 54, 55, 56, 57, 58]. While TAMP has demonstrated strong performance on long-horizon problems [59, 60, 61, 62, 63], they require symbolic representations of each skill's preconditions and effects, which are often unavailable or difficult to define in PNP settings. In particular, the NP skills typically involves multiple contact transitions and often yield stochastic outcomes where the object only approximately reaches the target pose, rendering defining effects difficult. Similar in spirit to our Skill-RRT, Barry et al. [64] propose an RRT-based skill planner that computes a collision-free object trajectory and identifies a sequence of skills to realize it. However, our method differs in three key ways: we use RL-based skills, we introduce connectors to bridge state gaps caused by the trained skills, and we reduce online computational time by distilling Skill-RRT into a reactive policy via IL.

Learning from planning solutions for manipulation Several works have proposed to generate IL data from planners. Driess et al. [65] generate TAMP datasets by using Logic-Geometric Programming (LGP), McDonald and Hadfield-Menell [66] uses FastForward for task planning and refine motion trajectories in simulators to train hierarchical policies that imitate both task and motion planning. OPTIMUS [30] uses PDDLStream [53] to train low-level policies. We take a similar approach but use Skill-RRT, a planner specifically designed for PNP problems. One core problem with planning data is that low-quality solutions can lead to failure during policy deployment [67]. OPTIMUS addresses this by filtering out trajectories where the end-effector exits a predefined workspace. While effective in simple pick-and-place domains, it is difficult to apply in contact-rich PNP problems. Instead, we evaluate the robustness of each plan by replaying it under stochastic disturbances in a simulator, keeping only plans that succeed under noise. Many works use a variety of different generative models to learn from multimodal planning solutions [5, 9, 30, 68, 69], such as Conditional Variational Auto Encoders and Gaussian Mixture Models. We use Diffusion Policy [5] which have proven its effectiveness in wide range of robotics applications.

3 Skill-Planning to Inference (SPIN)

At a high-level, SPIN uses Skill-RRT to collect skill plans, and then distill it to a policy using IL. But to find the skill plans using Skill-RRT, we first need connectors that allows chaining of

multiple skills, and to do that, we need `Lazy Skill-RRT` to collect relevant connector problems. See Figure 2 for the overview of `SPIN`. Let us begin by formally defining the PNP problem.

3.1 PNP manipulation problem formulation and `Skill-RRT`

We denote the state space as S, the action space as A. A state $s \in S$ includes object pose and velocity, robot joint positions, and velocities, denoted $q_{\text{obj}}, \dot{q}_{\text{obj}}, q_{\text{r}}$ and $\dot{q}_{\text{r}}$ respectively. An action $a \in A$ consists of the target end-effector pose, the gain and damping values for a low-level differential inverse kinematics (IK) controller, and the target gripper.

For each domain, we assume we are given a set of manipulation skills, $\mathcal{K} = \{K_{\text{P}}, K_{\text{NP}_1}, \cdots, K_{\text{NP}_n}\}$, such as a *topple* NP skill, and *pick-and-place* P skill. Each skill K is a tuple of two functions, $K = (\phi, \pi)$, where $\pi : S \times Q_{\text{obj}} \to A$ is a goal-conditioned policy trained using RL that maps a state s and a desired object pose q'_{obj} to an action, which is executed by a position controller, and $Q_{\text{obj}} \subset SE(3)$ is the space of stable object poses. The function $\phi : S \times Q_{\text{obj}} \to \{0, 1\}$ is an applicability checker that tests whether it is possible to execute the skill policy in s with $q_{\text{obj}} \in Q_{\text{obj}}$ as its goal, such as an IK solver that checks the existence of a feasible grasp for both the current and target object poses in a P skill. For more details about how the skill policies are structured and learned, and how ϕ are defined for each skill, see Appendix A.4, and A.5.

We assume we have a simulator f_{sim} that takes in state s and a policy conditioned on a desired object pose q'_{obj}, $\pi(\cdot; q'_{\text{obj}})$, and simulates the policy for N_{sim} number of time steps, and returns the next state, $f_{\text{sim}} : S \times \Pi \to S$ where Π is the set of policies of skills defined in $\mathcal{K}$. Given a pair of an initial state $s_0 \in S$ and the *ultimate* goal object pose $q^g_{\text{obj}} \in Q_{\text{obj}}$, the objective of a PNP problem is to find a sequence of skills and associated intermediate object poses, that we call a *skill plan*, $\tau_{\text{skill}} = \{\pi^{(t)}, q^{(t)}_{\text{obj}}\}_{t=1}^{H}$, such that when we simulate the sequence of H policies from s_0, the resulting state would have q^g_{obj} as its object pose. To solve this problem, we first generate training data using `Skill-RRT`, which we explain now.

Algorithm 1 `Skill-RRT`$(s_0, q^g_{\text{obj}}, \mathcal{K}, \mathcal{C}, Q_{\text{obj}})$

1: $T = \emptyset$
2: $v_0 \leftarrow \{(\emptyset, \emptyset), s_0\}$
3: T.`AddNode`$(parent = \emptyset, child = v_0)$
4: **for** $i = 1$ **to** N_{max} **do**
5: $\quad K, q'_{\text{obj}} \leftarrow$`UnifSmplSkillAndSubgoal`$(\mathcal{K}, Q_{\text{obj}})$
6: $\quad v_{\text{near}} \leftarrow$ `GetApplicableNearestNode`(T, K, q'_{obj})
7: $\quad$ **if** v_{near} is $\emptyset$ **then**
8: $\quad\quad$ **continue**
9: $\quad$`Extend`$(T, K, v_{\text{near}}, q'_{\text{obj}}, \mathcal{C})$
10: $\quad$ **if** `NearEnough`(q^g_{obj}, T) **then**
11: $\quad\quad$ **Return** `Retrace`(q^g_{obj}, T)
12: **Return** None

Algorithm 1 provides the pseudocode for `Skill-RRT`. The algorithm takes as input an initial state s_0, goal object configuration q^g_{obj}, a skill library $\mathcal{K}$, set of connectors $\mathcal{C}$, and object regions Q_{obj}. As we will soon explain, setting $\mathcal{C}$ to an empty set turns the algorithm into `Lazy Skill-RRT`. The algorithm begins by initiating the tree, T, with an empty set (L1), defining the root node (L2), and adding it to the tree (L3). A node v consists of a state s and a pair of a skill policy π and its desired object pose q'_{obj} that have been applied to the parent state to achieve the current state. For the root node, the policy and pose are set to an empty set (L2). We then begin the main for-loop. We first uniform-randomly sample a skill K and the desired object pose for the skill, q'_{obj}, using the function `UnifSmplSkillAndSubgoal` (L5), and compute the nearest node from the tree among the nodes where K can be applied (L6). Specifically, `GetApplicableNearestNode` function returns the nearest node v where $K.\phi(v.s, q_{\text{obj}})$ is true, and an empty set if no such node exists. If the function returns an empty set, we discard the sample and move to the next iteration (L7-8). Otherwise, we use `Extend` function with v_{near} (L9). Lastly, at every iteration, we check if q^g_{obj} is within a distance threshold ϵ from any node v's object pose in the tree, i.e., $\|q^g_{obj} - v.s.q_{obj}\| < \epsilon$. If the condition is satisfied, the `Retrace` function computes a sequence of vertices from the root node to that node (L10–11). If no such node can be found after N_{max} number of iterations, we return None (L12).

Algorithm 2 shows the `Extend` function. The function takes in the tree T, skill to be simulated K, node v that we will use K from, desired object pose we will extend to, q'_{obj}, and set of con-

nectors $\mathcal{C}$. The algorithm begins by computing a robot configuration from which K is applicable, q'_{r}, using the function `ComputePreSkillConfig` using state $v.s$ with q'_{obj} as a goal (L1). The difference between q'_{r} and the robot configuration in the current state, $v.s.q_{\text{r}}$, effectively defines the state gap that our connectors must fill. To do this, we create a *connecting node*, v_{connect} using `ComputeConnectingNode` shown in Algorithm 5 (Algorithm 2's L2). When connectors $\mathcal{C}$ is non-empty, this function simulates a connector and returns v_{connect} whose state is the result of simulating the connector. Otherwise, it returns a node whose state is the same as the current node's state $v.s$, but whose robot configuration is set to q'_{r}, to effectively define the problems for training the connectors. From the connecting node's state, $v_{\text{connect}}.s$, we simulate the policy $K.\pi$ with q'_{obj} as its goal (L3). If it fails to achieve q'_{obj}, then we return without modifying the tree (L4-5). Otherwise, we create a new node v' with the resulting state s', simulated policy $K.\pi$, and the desired object pose q'_{obj} (L6). We add v_{connect} with its parent as v, and v' with v_{connect} as its parent (L7-8). For all subrotuines that have been undefined, such as `ComputePreSkillConfig`, we provide their detailed pseudocodes and GPU-parallelized version used in our experiments in the Appendix B.1.

Algorithm 2 `Extend`$(T, K, v, q'_{\text{obj}}, \mathcal{C})$

1: $q'_{\text{r}} \leftarrow$ `ComputePreSkillConfig`$(K, v.s.q_{\text{obj}}, q'_{\text{obj}})$
2: $v_{\text{connect}} \leftarrow$ `ComputeConnectingNode`$(q'_{\text{r}}, K, \mathcal{C}, v)$
3: $s' \leftarrow f_{\text{sim}}(v_{\text{connect}}.s, K.\pi(v_{\text{connect}}.s; q'_{\text{obj}}))$
4: **if** `Failed`(s', q'_{obj}) **then**
5: **return**
6: $v' \leftarrow (K.\pi_{\text{post}}, q'_{\text{obj}}, s')$
7: T.`Add`$(parent = v, child = v_{\text{connect}})$
8: T.`Add`$(parent = v_{\text{connect}}, child = v')$

3.2 Training the connectors

To train a set of connectors $\mathcal{C}$, we collect problems using `Lazy Skill-RRT` by first creating a PNP problem that consists of a pair (s_0, q^g_{obj}), and then running Algorithm 1 with $\mathcal{C} = \emptyset$ in simulation. This will return a solution node sequence, some of which will be v_{connect}. We then collect a set of $(v.s, v_{\text{connect}}.s, K)$ triplet from the node sequence, where v is the parent of v_{connect}, and K is the skill that was used from v_{connect} to get its child node for which we will train a connector. We train a connector π_C whose goal is to go from $v.s.q_{\text{r}}$ to the connecting node's robot configuration $v_{\text{connect}}.s.q_{\text{r}}$, using PPO [32], with minimum disturbance to the object pose in $v.s$, for each K. The reward consists of four main components, $r_{\text{connector}} = r_{\text{ee}} + r_{\text{tip}} + r_{\text{obj-move}} + r_{\text{success}}$, where r_{ee} and r_{tip} are dense rewards based on the distances between the current and target end-effector poses and gripper tip positions, respectively. $r_{\text{obj-move}}$ penalizes an action if it changes the object pose from previous object pose, and r_{success} gives a large success reward for achieving the target end-effector pose and gripper width. More details about the implementation of the MDP formulation,and the hyperparameters for training the connectors, can be found in Appendix B.2. Having trained the connectors, now we can generate full solutions to PNP problems, which we use to train a policy.

3.3 Distilling `Skill-RRT` to a policy

While `Skill-RRT` can solve PNP problems, they are computationally expensive and must compute solutions from scratch for every initial and goal states. Therefore, `SPIN` distills the solutions given by `Skill-RRT` to train a policy using IL, so that it generalizes to different initial and goal states without an expensive tree search. To generate data, we solve a set of PNP problems using `Skill-RRT`, and then collect a data of the form (s, a), where a is given by each skill policy used in the skill plan, and s is the state encountered during the execution of a skill plan, and we use it to train a policy. To facilitate zero-shot real-world deployment, we use a slightly different state representation from the ones defined previously. See Appendix B.3 for details.

Distilling a planner to a policy requires solving two challenges: (1) selecting high-quality skill plans for distillation, and (2) choosing the appropriate learning algorithm that accounts for the characteristics of our data. For (1), for each skill plan, we replay it N times with Gaussian or uniform random noise injected into states and joint torques, and randomize physical parameters such as friction and mass. Plans with a success rate above a threshold m, i.e., $\frac{N_{success}}{N} > m$, are retained. This process ensures that the dataset remains robust to perception and modeling errors and focus on solutions

likely to reach the goal despite these disturbances and uncertainties. For details about the noise we used, see Appendix B.4. For (2), even for the same problem (i.e., the same s_0 and q^g_{obj}), skill plans will have different intermediate subgoals resulting in multi-modal solutions. To handle such multi-modality, we use Diffusion Policy [5] as our policy. Details about the policy architecture and training hyperparameters can be found in Appendix B.3.

4 Experiments

4.1 Simulation results

We evaluate our method in three domains: *Card Flip*, *Bookshelf*, and *Kitchen* (Figure 1). Each domain is equipped with a set of skills $\mathcal{K}$. In Card Flip, we have $\mathcal{K} = \{K_{\text{slide}}, K_{\text{prehensile}}\}$, where K_{slide} slides the card on the table. In Bookshelf, $\mathcal{K} = \{K_{\text{topple}}, K_{\text{push}}, K_{\text{prehensile}}\}$, where K_{topple} topples a book on the upper shelf, and K_{push} pushes a book on the lower shelf. In Kitchen, $\mathcal{K}_{\text{kitchen}} = \{K_{\text{sink}}, K_{\text{cupboard}}, K_{\text{prehensile}}\}$, where K_{sink} and K_{cupboard} are non-prehensile skills that manipulates objects to the target pose in the sink and the cupboard respectively. In all domains, $K_{\text{prehensile}}$ is a pick-and-place skill that moves an object from its current pose to the target pose. We choose initial and goal object poses such that no single skill suffices, requiring combination of multiple skills to reach the goal. See Appendix A.1 for details of our problem distribution.

We compare our method with the following baselines:

- PPO [32]: An RL method that directly outputs joint torque commands along with joint gain and damping values without utilizing pre-trained skills $\mathcal{K}$. Trained with a reward function based on the distance between the current and goal object poses.
- `HLPS` [31]: A state-of-the-art hierarchical RL method that, instead of using $\mathcal{K}$, simultaneously learns low-level and high-level policies to achieve the goal. The high-level policy outputs a subgoal for a low-level policy, and is trained with the reward function based on the distance between the current and goal object pose. The low-level policy is trained with the reward function based on the distance to the subgoal provided by the high-level policy.
- `MAPLE` [24]: A state-of-the-art PAMDP RL method trained with a reward function that rewards the robot when the high-level policy selects a feasible skill, successfully completes the task, or reduces the distance between the current and goal object poses. Uses pre-trained skills $\mathcal{K}$.
- `Skill-RRT`: Our planner without policy distillation.
- `SPIN`: Our method with the replay success threshold of $m = 0.9$
- `SPIN-w/o-filtering`: Our method without data filtering

The summary of these baselines is presented in the left of Table 1. The details for how we train these baselines and their MDP definitions are included in Appendix C.1.1, C.1.2, and C.1.3. For each baseline, we report the number of state-action pairs used for training across the three domains in Appendix Table 24. We use two metrics to evaluate the baselines: success rate (# success episodes / # problems attempted) and computation time. For `Skill-RRT`, this is the average amount of plan computation time, and for all other methods, this is the sum of all inference times during the episode, averaged over all successful episodes. We use 100 problems to evaluate baselines.

Table 1 (right) shows the results. PPO, which does not use temporally-abstracted actions, achieves zero success rate across all domains, because the tasks are too long-horizon (up to 564 number of low-level actions) for algorithms with a flat action space. `HLPS`, which simultaneously learns to do the task and discover low-level policies, also achieves zero success in all domains because of the fundamental problem with hierarchical RL: the high-level policy must adapt to the evolving low-level policy, and the training is inherently unstable. `MAPLE`, on the other hand, leverages the pre-trained skills $\mathcal{K}$, and achieves 78% in the bookshelf domain. However, in two other domains where there is a very small region of intermediate object poses that the robot must go through to achieve the goal, such as the edge of the table for Card Flip, it achieves zero success rates (see Appendix Figure 7 for a visual illustration). In contrast, `Skill-RRT` achieves 39%, 66%, and 64% success rates in the three domains. This is because it (1) leverages pre-trained skills, $\mathcal{K}$, and (2) uses sophisticated exploration strategy based on Voronoi Bias [70] to find a path through a narrow

	Components			Problem Domain					
				Card Flip		Bookshelf		Kitchen	
	Method	Action Type	Use $\mathcal{K}$	Success rate (%)	Computation time (s)	Success rate (%)	Computation time (s)	Success rate (%)	Computation time (s)
PPO [32]	Flat RL	Low-level action	✗	0.0	N/A	0.0	N/A	0.0	N/A
HLPS [31]	Hierarchical RL	Low-level action	✗	0.0	N/A	0.0	N/A	0.0	N/A
MAPLE [24]	Hierarchical RL	Skill & Parameter	✓	0.0	N/A	78.0	5.3	0.0	N/A
Skill-RRT	Planning	Skill & Parameter	✓	39.0	85.3 ± 48.7	66.0	79.2 ± 67.1	64.0	121 ± 39.5
SPIN-w/o-filtering	Planner distilled via IL	Low-level action	✓	82.0	2.68 ± 0.61	83.0	2.93 ± 2.05	87.0	3.02 ± 0.65
SPIN	Planner distilled via IL	Low-level action	✓	**95.0**	2.68 ± 0.61	**93.0**	2.93 ± 2.05	**98.0**	3.02 ± 0.65

Table 1: Description of baselines (Left) and performance metrics (Right) with three different random seeds for each domain (standard deviations are reported in Table 25).

passage. However, because of narrow passages, its average plan computation time is 85.3s, 69.2s, and 121s for the three domains, which is impractical for a real-world deployment.

In contrast, both variants of SPIN achieve the highest success rates and speed among all baselines. This is because the distilled policy is trained on a diverse set of successful skill plans generated by Skill-RRT, which allows it to generalize across states and recover from mistakes, leading to improved performance compared to Skill-RRT that executes its plan directly, without a feedback loop. We also see the impact of our data filtering scheme: SPIN has about 10 percent improvement in success rates compared to SPIN-w/o-filtering. This is because our noise-injection-based filtering scheme eliminates skill plans that are sensitive to small deviations, such as placing the card very near the edge of the table in which even a slight error during execution would lead to failure. By filtering out such data, the policy is distilled from more robust and reproducible skill sequences, resulting in improved execution stability.

We conduct additional ablation studies on three aspects: (1) the necessity of learning the connectors, where we compare our learned connectors against the motion planner-based baseline (see supplementary video[‡]); (2) the importance of the data filtering method during imitation learning, supported by qualitative analysis; and (3) the impact of different model architectures on imitation learning performance (e.g., GMM, VAE). Detailed results and analyses are provided in Appendix D.

4.2 Real World Experiments

We evaluate our distilled policy in the real world by zero-shot transferring the policy trained in simulation. For each domain, we use a set of 20 test problems consisting of different initial and goal states, and the robot must solve them in real time. The shapes of the real environments and objects are identical to those in simulation. Our policy achieves 85% (17/20), 90% (18/20) and 80% (16/20) in the card flip, bookshelf, and kitchen domains respectively. There are nine failures in total, and their causes can be categorized into three cases. The first is unexpected contacts between the robot and the object during NP skill execution, which drive the object into out-of-distribution states (55.5%, 5/9). In the Kitchen domain, for example, an unintended collision between the cup and the robot causes the cup to become trapped against the sink wall, preventing the robot to make further progress. The second is the hardware torque limit violations, caused by a strong collision with the environment during NP skill execution (22.2%, 2/9). The third is geometric collisions between the object and the environment (22.2%, 2/9), such as book colliding with the lower shelf while the robot tries to pick-and-place it to the lower shelf in the Bookshelf domain.

5 Conclusion

In this work, we present SPIN, consisting of: (1) Skill-RRT, a planner that extends RRT for PNP problems by discovering sequences of skills and subgoals albeit at high computational cost, (2) a way to close the state gap between skills using connectors, (3) and a noise-based data filtering and diffusion-based IL algorithms for distilling Skill-RRT into a policy. We show its real-world experiments demonstrating the effectiveness of our framework in three challenging domains.

[‡]Video link: https://www.youtube.com/watch?v=Z2I_DoPOuX8

6 Limitations

There are several limitations to our work. First, `SPIN` successfully generalizes over initial and goal object poses, but does not generalize over object and environment shapes. We deem this for future work.

In terms of scalability and structural limitations, although our algorithm can automatically generate high-quality skill plans (manipulation demonstrations), the data collection process remains time-consuming, and the computation cost increases with the number of skills and planning regions. We anticipate that advances in simulation and computing hardware will help alleviate this issue.

A more fundamental limitation lies in our algorithm `Lazy Skill-RRT`: the connecting node v_{connect} is constructed under the assumption that the object's configuration remains unchanged from the nearest node v. This restricts the framework to skills with minimal terminal object motion, and prevents the use of dynamic skills such as throwing or batting. In practice, this limitation is not critical for our current tasks, as most household manipulation skills involve low-velocity object motion. Furthermore, dynamic skills are not yet supported by our simulation, and current real-world robotic platforms lack the mechanical bandwidth and control precision required for accurate execution of such fast, contact-rich behaviors.

Acknowledgments

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant and National Research Foundation of Korea (NRF) funded by the Korea government(MSIT) (No.2019-0-00075, Artificial Intelligence Graduate School Program(KAIST)), (No.2022-0- 00311, Development of Goal-Oriented Reinforcement Learning Techniques for Contact-Rich Robotic Manipulation of Everyday Objects), (No. 2022-0-00612, Geometric and Physical Commonsense Reasoning based Behavior Intelligence for Embodied AI), (No. RS-2024-00359085, Foundation model for learning-based humanoid robot that can understand and achieve language commands in unstructured human environments), (No. RS-2024-00509279, Global AI Frontier Lab).

References

[1] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on robot learning*, pages 651–673. PMLR, 2018.

[2] A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, B. Sundaralingam, et al. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5977–5984. IEEE, 2023.

[3] Y. Cho, J. Han, Y. Cho, and B. Kim. Corn: Contact-based object representation for nonprehensile manipulation of general unseen objects. *arXiv preprint arXiv:2403.10760*, 2024.

[4] T. Chen, M. Tippur, S. Wu, V. Kumar, E. Adelson, and P. Agrawal. Visual dexterity: In-hand reorientation of novel and complex object shapes. *Science Robotics*, 8(84):eadc9244, 2023.

[5] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[6] Z. Fu, T. Z. Zhao, and C. Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117*, 2024.

[7] M. Reuss, M. Li, X. Jia, and R. Lioutikov. Goal-Conditioned Imitation Learning using Score-based Diffusion Policies. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi:10.15607/RSS.2023.XIX.028.

[8] C. Wang, L. Fan, J. Sun, R. Zhang, L. Fei-Fei, D. Xu, Y. Zhu, and A. Anandkumar. Mimicplay: Long-horizon imitation learning by watching human play. In *Conference on Robot Learning*, pages 201–221. PMLR, 2023.

[9] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi:10.15607/RSS.2023.XIX.016.

[10] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, et al. What matters for on-policy deep actor-critic methods? a large-scale study. In *International conference on learning representations*, 2021.

[11] C. Chi, Z. Xu, C. Pan, E. Cousineau, B. Burchfiel, S. Feng, R. Tedrake, and S. Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. *arXiv preprint arXiv:2402.10329*, 2024.

[12] T. Lin, Y. Zhang, Q. Li, H. Qi, B. Yi, S. Levine, and J. Malik. Learning visuotactile skills with two multifingered hands. *arXiv preprint arXiv:2404.16823*, 2024.

[13] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. Aparicio, and K. Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *Robotics: Science and Systems XIII*, 2017.

[14] S. Joshi, S. Kumra, and F. Sahin. Robotic grasping using deep reinforcement learning. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 1461–1466. IEEE, 2020.

[15] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13438–13444. IEEE, 2021.

[16] H.-S. Fang, C. Wang, H. Fang, M. Gou, J. Liu, H. Yan, W. Liu, Y. Xie, and C. Lu. Anygrasp: Robust and efficient grasp perception in spatial and temporal domains. *IEEE Transactions on Robotics*, 2023.

[17] W. Zhou and D. Held. Learning to grasp the ungraspable with emergent extrinsic dexterity. In *Conference on Robot Learning*, pages 150–160. PMLR, 2023.

[18] M. Kim, J. Han, J. Kim, and B. Kim. Pre-and post-contact policy decomposition for non-prehensile manipulation with zero-shot sim-to-real transfer. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10644–10651. IEEE, 2023.

[19] W. Zhou, B. Jiang, F. Yang, C. Paxton, and D. Held. Hacman: Learning hybrid actor-critic maps for 6d non-prehensile manipulation. *arXiv preprint arXiv:2305.03942*, 2023.

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

[21] 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.

[22] B. Li, H. Tang, Y. Zheng, J. Hao, P. Li, Z. Wang, Z. Meng, and L. Wang. Hyar: Addressing discrete-continuous action reinforcement learning via hybrid action representation. *arXiv preprint arXiv:2109.05490*, 2021.

[23] M. Dalal, D. Pathak, and R. R. Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. *Advances in Neural Information Processing Systems*, 34: 21847–21859, 2021.

[24] S. Nasiriany, H. Liu, and Y. Zhu. Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 7477–7484. IEEE, 2022.

[25] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, S. Sorkin, et al. On finding narrow passages with probabilistic roadmap planners. In *Robotics: the algorithmic perspective: 1998 workshop on the algorithmic foundations of robotics*, pages 141–154, 1998.

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

[27] S. LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

[28] G. Konidaris and A. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in neural information processing systems*, 22, 2009.

[29] M. Mason. The mechanics of manipulation. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 544–548. IEEE, 1985.

[30] M. Dalal, A. Mandlekar, C. R. Garrett, A. Handa, R. Salakhutdinov, and D. Fox. Imitating task and motion planning with visuomotor transformers. In *Conference on Robot Learning*, pages 2565–2593. PMLR, 2023.

[31] V. H. Wang, T. Wang, W. Yang, J.-K. Kämäräinen, and J. Pajarinen. Probabilistic subgoal representations for hierarchical reinforcement learning. *arXiv preprint arXiv:2406.16707*, 2024.

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[33] L. Wang, Y. Xiang, W. Yang, A. Mousavian, and D. Fox. Goal-auxiliary actor-critic for 6d robotic grasping with point clouds. In *Conference on Robot Learning*, pages 70–80. PMLR, 2022.

[34] J. D. A. Ferrandis, J. Moura, and S. Vijayakumar. Nonprehensile planar manipulation through reinforcement learning with multimodal categorical exploration. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5606–5613. IEEE, 2023.

[35] W. Yuan, J. A. Stork, D. Kragic, M. Y. Wang, and K. Hang. Rearrangement with nonprehensile manipulation using deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 270–277. IEEE, 2018.

[36] W. Yuan, K. Hang, D. Kragic, M. Y. Wang, and J. A. Stork. End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer. *Robotics and Autonomous Systems*, 119:119–134, 2019.

[37] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

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

[39] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[40] A. Harutyunyan, W. Dabney, D. Borsa, N. Heess, R. Munos, and D. Precup. The termination critic. *arXiv preprint arXiv:1902.09996*, 2019.

[41] U. Singh and V. P. Namboodiri. Pear: Primitive enabled adaptive relabeling for boosting hierarchical reinforcement learning. *arXiv preprint arXiv:2306.06394*, 2023.

[42] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

[43] M. Hutsebaut-Buysse, K. Mets, and S. Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1):172–221, 2022.

[44] O. Nachum, S. S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[45] T. Zhang, S. Guo, T. Tan, X. Hu, and F. Chen. Adjacency constraint for efficient hierarchical reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45 (4):4152–4166, 2022.

[46] A. Franceschetti, E. Tosello, N. Castaman, and S. Ghidoni. Robotic arm control and task training through deep reinforcement learning. In *International Conference on Intelligent Autonomous Systems*, pages 532–550. Springer, 2021.

[47] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[48] H. Wang, H. Zhang, L. Li, Z. Kan, and Y. Song. Task-driven reinforcement learning with action primitives for long-horizon manipulation skills. *IEEE Transactions on Cybernetics*, 54 (8):4513–4526, 2023.

[49] H. Fu, H. Tang, J. Hao, Z. Lei, Y. Chen, and C. Fan. Deep multi-agent reinforcement learning with discrete-continuous hybrid action spaces. *arXiv preprint arXiv:1903.04959*, 2019.

[50] E. Wei, D. Wicke, and S. Luke. Hierarchical approaches for reinforcement learning in parameterized action space. In *AAAI Spring Symposia*, 2018.

[51] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4(1):265–293, 2021.

[52] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 37(1): 104–136, 2018.

[53] 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*, volume 30, pages 440–448, 2020.

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

[55] T. Ren, G. Chalvatzaki, and J. Peters. Extended tree search for robot task and motion planning. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12048–12055. IEEE, 2024.

[56] T. Migimatsu and J. Bohg. Object-centric task and motion planning in dynamic environments. *IEEE Robotics and Automation Letters*, 5(2):844–851, 2020.

[57] M. Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *IJCAI*, pages 1930–1936, 2015.

[58] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. 2018.

[59] Y. Du, M. Yang, P. Florence, F. Xia, A. Wahid, B. Ichter, P. Sermanet, T. Yu, P. Abbeel, J. B. Tenenbaum, et al. Video language planning. *arXiv preprint arXiv:2310.10625*, 2023.

[60] B. Kim, L. Shimanuki, L. P. Kaelbling, and T. Lozano-Pérez. Representation, learning, and planning algorithms for geometric task and motion planning. *The International Journal of Robotics Research*, 41(2):210–231, 2022.

[61] J. Mendez-Mendez, L. P. Kaelbling, and T. Lozano-Pérez. Embodied lifelong learning for task and motion planning. In *Conference on Robot Learning*, pages 2134–2150. PMLR, 2023.

[62] B. Vu, T. Migimatsu, and J. Bohg. Coast: Constraints and streams for task and motion planning. *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[63] Y. Zhu, J. Tremblay, S. Birchfield, and Y. Zhu. Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6541–6548. IEEE, 2021.

[64] J. Barry, L. P. Kaelbling, and T. Lozano-Pérez. A hierarchical approach to manipulation with diverse actions. In *2013 IEEE International Conference on Robotics and Automation*, pages 1799–1806. IEEE, 2013.

[65] D. Driess, J.-S. Ha, R. Tedrake, and M. Toussaint. Learning geometric reasoning and control for long-horizon tasks from visual input. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 14298–14305. IEEE, 2021.

[66] M. J. McDonald and D. Hadfield-Menell. Guided imitation of task and motion planning. In *Conference on Robot Learning*, pages 630–640. PMLR, 2022.

[67] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning*, pages 1678–1690. PMLR, 2022.

[68] S. Lee, Y. Wang, H. Etukuru, H. J. Kim, N. M. M. Shafiullah, and L. Pinto. Behavior generation with latent actions. *arXiv preprint arXiv:2403.03181*, 2024.

[69] N. M. Shafiullah, Z. Cui, A. A. Altanzaya, and L. Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.

[70] S. R. Lindemann and S. M. LaValle. Incrementally reducing dispersion by increasing voronoi bias in rrts. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 4, pages 3251–3257. IEEE, 2004.

[71] W. B. Powell. Ai, or and control theory: A rosetta stone for stochastic optimization. *Princeton University*, page 12, 2012.

[72] C. Eppner, A. Mousavian, and D. Fox. Acronym: A large-scale grasp dataset based on simulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6222–6227. IEEE, 2021.

[73] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[74] B. Wen, W. Yang, J. Kautz, and S. Birchfield. Foundationpose: Unified 6d pose estimation and tracking of novel objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17868–17879, 2024.

[75] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023.

[76] H. K. Cheng, S. W. Oh, B. Price, J.-Y. Lee, and A. Schwing. Putting the object back into video object segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3151–3161, 2024.

[77] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi:10.1109/MRA.2012.2205651. https://ompl.kavrakilab.org.

[78] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[79] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning (CoRL)*, 2021.

[80] D. P. Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[81] A. Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

A Problem Details

In this section, we provide a detailed description of the problem setup used throughout our framework. Specifically, we describe (1) how problems, defined as pairs of initial states and goal object poses, are generated; (2) how regions are defined in each domain; (3) which skills are assigned to each domain; and (4) the training and structure of the non-prehensile and prehensile skill policies in the skill library $\mathcal{K}$.

A.1 Problem Generation

Each PNP problem is defined by a pair consisting of an initial state s_0 and a goal object pose q^g_{obj}. The specifics of the initial state and goal object pose vary based on the environment, and this appendix provides a detailed explanation of their construction.

To simplify the problem setup, it is assumed that both the robot and the object are stationary at the initial state. As a result, the initial velocities are set to $s_0.\dot{q}_r = 0$ and $s_0.\dot{q}_{\text{obj}} = 0$. The initial joint positions of the robot $s_0.q_r$ are sampled randomly, but this sampling is constrained to satisfy two conditions: the positions must lie within the robot's joint limits, and there must be no collisions with the surrounding environment, including obstacles such as tables, bookshelves, kitchen furniture, or objects. The sampling of the initial object pose $s_0.q_{\text{obj}}$ and the goal object pose q^g_{obj} is performed from *distinct regions* within the environment-specific sampling spaces, ensuring that the initial and goal poses are drawn from separate regions. In particular, we intentionally sample object poses for which no collision-free and graspable robot configuration exists, i.e., there does not exist any robot configuration that can reach the object without collision and enable a feasible grasp at the sampled object pose. This ensures that the task requires, at a minimum, the use of a non-prehensile skill to reposition the object before applying a prehensile skill, followed by another non-prehensile skill if necessary. The detailed definitions of the sampling spaces for the initial and goal object poses are provided below.

Card Flip: The initial object pose, $s_0.q_{\text{obj}}$, is sampled such that the object is fully positioned within the boundaries of the table, regardless of whether it is flipped or unflipped. Similarly, the goal object pose, q^g_{obj}, is sampled entirely within the table boundaries but must have the opposite orientation from the initial object pose (i.e., if $s_0.q_{\text{obj}}$ is unflipped, then q^g_{obj} must be flipped, and vice versa).

- $s_0.q_{\text{obj}}$: The initial object pose is sampled from $\{(x, y, z, \theta_x, \theta_y, \theta_z) \mid x \in [x^{\text{table}}_{min}, x^{\text{table}}_{max}], y \in [y^{\text{table}}_{min}, y^{\text{table}}_{max}], z = z^{\text{table}}, \theta_x \in \{0, \pi\}, \theta_y = 0, \theta_z \in [0, 2\pi]\}$.
- q^g_{obj}: The goal object pose is sampled from $\{(x, y, z, \theta_x, \theta_y, \theta_z) \mid x \in [x^{\text{table}}_{min}, x^{\text{table}}_{max}], y \in [y^{\text{table}}_{min}, y^{\text{table}}_{max}], z = z^{\text{table}}, \theta_x = s_0.q_{\text{obj}}.\theta_x + \pi, \theta_y = 0, \theta_z \in [0, 2\pi]\}$.

Bookshelf: The initial object pose, $s_0.q_{\text{obj}}$, is sampled on the upper bookshelf, with the book placed upright and inserted into the shelf. The goal object pose, q^g_{obj}, is sampled on the lower bookshelf, positioned fully inside the shelf for storage.

- $s_0.q_{\text{obj}}$: The initial object pose is sampled from $\{(x, y, z, \theta_x, \theta_y, \theta_z) \mid x = x^{\text{upper-shelf}}, y \in [y^{\text{upper-shelf}}_{min}, y^{\text{upper-shelf}}_{max}], z = z^{\text{upper-shelf}}, \theta_x = \pi/2, \theta_y = 0, \theta_z = 0\}$.
- q^g_{obj}: The goal object pose is sampled from $\{(x, y, z, \theta_x, \theta_y, \theta_z) \mid x = x^{\text{lower-shelf}}, y \in [y^{\text{lower-shelf}}_{min}, y^{\text{lower-shelf}}_{max}], z = z^{\text{lower-shelf}}, \theta_x = 0, \theta_y = 0, \theta_z = 0\}$.

Kitchen: The initial object pose, $s_0.q_{\text{obj}}$, is sampled within the sink. The goal object pose, q^g_{obj}, is sampled on the upper shelf, which can be either the left upper shelf or the right upper shelf, selected randomly.

- $s_0.q_{\text{obj}}$: Unlike objects such as cards or books, due to the asymmetry geometry of the cup, we sample $s_0.q_{\text{obj}}$ from a precollected dataset containing initial poses in the sink. This dataset is generated by dropping the cup into the sink from a sufficient height and recording its initial pose in simulation when it's lying down.
- q^g_{obj}: The goal object pose is sampled from $\{(x, y, z, \theta_x, \theta_y, \theta_z) \mid x = x^{\text{cupboard}}, y \in \{y^{\text{l-cupboard}}, y^{\text{r-cupboard}}\}, z = z^{\text{cupboard}}, \theta_x = 0, \theta_y = 0, \theta_z \in [5\pi/6, 7\pi/6]\}$.

For each problem, both the initial object pose, $s_0.q_{\text{obj}}$, and the goal object pose, q^g_{obj}, are randomly sampled.

A.2 Set of Domain Regions

One well-known problem when planning with parameterized manipulation skills is that if you sample the object pose q_{obj} from the full $SE(3)$ space, the probability of sampling a stable pose is zero, and planning becomes intractable [51]. Therefore, following prior work [51], we associate each distinct region of stable poses with a NP skill, while the region for a P skill is defined as the union of these regions, which corresponds to Q_{obj}. For example, in the bookshelf domain, the upper shelf is associated with topple or push skills, the lower shelf with the push skill, and the P skill enables moving the book between these regions.

To make planning feasible and to define the subgoal space for skill-based baselines such as `MAPLE`, we define Q_{obj} as a subset of the $SE(3)$ space where the object can be placed stably. In the card flip domain, Q_{obj} includes object poses lying flat on the table, regardless of whether the card is flipped. In the bookshelf domain, Q_{obj} consists of two types: $Q^{\text{upper-shelf}}_{\text{obj}}$, where the book is fitted between other books on the upper shelf, and $Q^{\text{lower-shelf}}_{\text{obj}}$, where the book is lying on the lower shelf. In the kitchen domain, Q_{obj} includes $Q^{\text{sink}}_{\text{obj}}$ (cup in the sink), $Q^{\text{l-cupboard}}_{\text{obj}}$ (cup on the left side of the cupboard), and $Q^{\text{r-cupboard}}_{\text{obj}}$ (cup on the right side). Figure 3 illustrates examples of Q_{obj} in each domain.

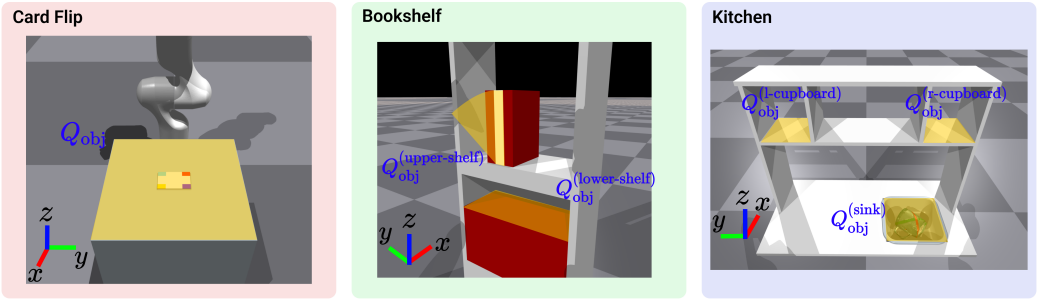

Figure 3: Regions for each domain.

A.3 Skills For Each Domain

In this section, we describe how the skill library $\mathcal{K}$ is constructed for each domain, and introduce the structure of each skill K. As outlined in Table 2 and Table 3, each skill K consists of two components: (1) an applicability checker ϕ, and (2) a goal-conditioned policy π.

Card Flip domain:

The set of skills $\mathcal{K}$ consists of two different skills: $\{K_{\text{NP}_{\text{slide}}}, K_{\text{P}}\}$.

- $K_{\text{NP}_{\text{slide}}}$: A NP skill that manipulates a card by sliding it between two poses within the region Q_{obj}, ensuring that the card's orientation remains unchanged along the global x- and y-axes.

Domain names	Card Flip	Bookshelf		Kitchen	
NP skills	K_{slide}	K_{topple}	K_{push}	K_{sink}	K_{cupboard}
$\phi(q, q')$	$q, q' \in Q_{\text{obj}}$, $R_x(q) = R_x(q')$, $R_y(q) = R_y(q')$	$q, q' \in Q_{\text{obj}}^{\text{(upper-shelf)}}$, $R_x(q) = R_x(q')$, $R_z(q) = R_z(q')$	$q, q' \in Q_{\text{obj}}^{\text{(lower-shelf)}}$, $R_x(q) = R_x(q')$, $R_y(q) = R_y(q')$	$q, q' \in Q_{\text{obj}}^{\text{(sink)}}$,	$q, q' \in Q_{\text{obj}}^{\text{(l-cupboard)}}$ or $Q_{\text{obj}}^{\text{(r-cupboard)}}$, $R_x(q) = R_x(q')$, $R_y(q) = R_y(q')$
π	π_{slide}	π_{topple}	π_{push}	π_{sink}	π_{cupboard}

Table 2: Non-prehensile manipulation skills trained using RL for each domain. The row $\phi(q, q')$ denotes the applicability checker for each skill. We write $q = q_{\text{obj}}$ and $\phi(q, q')$ instead of $\phi(s, q')$ with abuse of notation for brevity and clarity. Here, R_x, R_y and R_z denote the rotation matrices of pose q with respect to x, y, and z axes. For the card flip domain, slide skill is applicable if the desired pose q' involves only translation and rotation wrt the z-axis from q. For the bookshelf domain, we have two sub-regions $Q_{\text{obj}}^{\text{(upper-shelf)}}$ and $Q_{\text{obj}}^{\text{(lower-shelf)}}$, as shown in Figure 3 (middle). To apply the topple or push skill, q and q' must belong to the same sub-region. Toppling skill is applicable only for orientation wrt the y-axis. The pushing skill is applicable for orientation wrt the z-axis and any translation within the lower shelf. For the kitchen domain, we have three sub-regions: $Q_{\text{obj}}^{\text{(sink)}}$, $Q_{\text{obj}}^{\text{(l-cupboard)}}$, and $Q_{\text{obj}}^{\text{(r-cupboard)}}$. K_{sink} is applicable for any orientation or translation, as long as the object stays within the sink. K_{cupboard} is applicable for an orientation wrt z-axis and any translation, as long as the object moves within a left or right cupboard. The last row, π, indicates different policies trained for different skills. The details of training are in Appendix A.4.

Domain names	Card Flip	Bookshelf	Kitchen
P skills	K_{P}	K_{P}	K_{P}
$\phi(q, q')$	$q, q' \in Q_{\text{obj}}, \quad \mathcal{G}(q) \cap \mathcal{G}(q') \neq \emptyset$		
π	$\pi_{\text{P}}^{\text{Card-Flip}}$	$\pi_{\text{P}}^{\text{Bookshelf}}$	$\pi_{\text{P}}^{\text{Kitchen}}$

Table 3: Prehensile manipulation skills trained using RL for each domain. The row $\phi(q, q')$ denotes the applicability checker for each skill. We write $q = q_{\text{obj}}$ and $\phi(q, q')$ instead of $\phi(s, q')$ with slight abuse of notation for brevity and clarity. Specifically, $q, q' \in Q_{\text{obj}}$ and a skill is applicable if $\mathcal{G}(q) \cap \mathcal{G}(q') \neq \emptyset$, where $\mathcal{G}(q)$ denotes the set of feasible grasps at object pose q: $\mathcal{G}(q) = \{g \in \mathcal{G}_{\text{predef}}(q) \mid \text{IK}(g) \text{ is feasible}\}$. Here, $\mathcal{G}_{\text{predef}}(q)$ represents the set of predefined grasps relative to the object pose of q, and $\text{IK}(g)$ indicates whether the grasp g admits a valid inverse kinematics solution. The last row, π, indicates different policies trained for different skills. Policy details and additional descriptions of $\mathcal{G}_{\text{predef}}(q)$ are provided in Appendix A.5.

  - $K_{\text{NP}_{\text{slide}}}.\phi(q_{\text{obj}}, q'_{\text{obj}})$ determines whether an object can slide from q_{obj} to q'_{obj}. It returns 1 if both poses belong to Q_{obj}, and if both q_{obj} and q'_{obj} have identical orientations with respect to the global x- and y-axes.

- K_{P}: A P skill that moves a card within Q_{obj} while maintaining a grasp on it.
  - $K_{\text{P}}.\phi(q_{\text{obj}}, q'_{\text{obj}})$ returns 1 if $q, q' \in Q_{\text{obj}}$ and there exists at least one common feasible grasp between q and q', that is, if $\mathcal{G}(q) \cap \mathcal{G}(q') \neq \emptyset$. Otherwise, it returns 0. Here, $\mathcal{G}(q) = \{g \in \mathcal{G}_{\text{predef}}(q) \mid \text{IK}(g) \text{ is feasible}\}$ denotes the set of predefined grasps at pose q that admit a valid inverse kinematics solution. The predefined grasp set $\mathcal{G}_{\text{predef}}(q)$ is defined for the card object in the card flip domain.

<u>Bookshelf domain:</u>

The set of skills $\mathcal{K}$ consists of three different skills: $\{K_{\text{NP}_{\text{topple}}}, K_{\text{NP}_{\text{push}}}, K_{\text{P}}\}$.

- $K_{\text{NP}_{\text{topple}}}$: A NP skill that transitions a book from an upright pose to a toppled pose within the upper bookshelf region $Q_{\text{obj}}^{\text{upper-shelf}}$, ensuring that the book maintains its alignment along the global x- and z-axes.
  - $K_{\text{NP}_{\text{topple}}}.\phi(q_{\text{obj}}, q'_{\text{obj}})$ determines whether an object on the upper shelf can transition between poses while toppling. It returns 1 if both poses belong to $Q_{\text{obj}}^{\text{upper-shelf}}$ and if both q_{obj} and q'_{obj} have identical orientations with respect to the global x- and z-axes.

- $K_{\text{NP}_{\text{push}}}$: A NP skill that moves a book deeper into the lower bookshelf region $Q_{\text{obj}}^{\text{lower-shelf}}$ by pushing it, ensuring that the book retains its orientation along the global x- and y-axes.

- $K_{\text{NP}_{\text{push}}}.\phi(q_{\text{obj}}, q'_{\text{obj}})$ determines whether an object on the lower shelf can be pushed. It returns 1 if both poses belong to $Q_{\text{obj}}^{\text{lower-shelf}}$, and if both q_{obj} and q'_{obj} have identical orientations with respect to the global x- and y-axes.

- K_{P}: A P skill that moves a book from the upper bookshelf to the lower bookshelf while maintaining a grasp on it.
  - $K_{\text{P}}.\phi(q_{\text{obj}}, q'_{\text{obj}})$ is identical to $\phi_{\text{P}}(q_{\text{obj}}, q'_{\text{obj}})$ in the card domain, except that the predefined grasp set $\mathcal{G}_{\text{predef}}(q)$ is defined for the book object instead of the card.

<u>Kitchen domain:</u>

The set of skills $\mathcal{K}$ consists of four different skills: $\{K_{\text{NP}_{\text{sink}}}, K_{\text{NP}_{\text{l-cupboard}}}, K_{\text{NP}_{\text{r-cupboard}}}, K_{\text{P}}\}$.

- $K_{\text{NP}_{\text{sink}}}$: A NP skill that manipulates a cup between two poses within the sink region $Q_{\text{obj}}^{\text{sink}}$, ensuring that the cup remains within the boundaries of the sink throughout the movement.
  - $K_{\text{NP}_{\text{sink}}}.\phi(q_{\text{obj}}, q'_{\text{obj}})$ determines whether an object on the sink can be manipulated. It returns 1 if both poses belong to $Q_{\text{obj}}^{\text{sink}}$.

- $K_{\text{NP}_{\text{l-cupboard}}}$: A NP skill that moves a cup within the region $Q_{\text{obj}}^{\text{l-cupboard}}$, ensuring that the cup's orientation remains unchanged along the global x- and y-axes.
  - $K_{\text{NP}_{\text{l-cupboard}}}.\phi(q_{\text{obj}}, q'_{\text{obj}})$ determines whether an object on the left cupboard can be pushed. It returns 1 if both poses belong to $Q_{\text{obj}}^{\text{l-cupboard}}$, and if both q_{obj} and q'_{obj} have identical orientations with respect to the global x- and y-axes.

- $K_{\text{NP}_{\text{r-cupboard}}}$: A NP skill that moves a cup within the region $Q_{\text{obj}}^{\text{r-cupboard}}$, ensuring that the cup's orientation remains unchanged along the global x- and y-axes.
  - $K_{\text{NP}_{\text{r-cupboard}}}.\phi(q_{\text{obj}}, q'_{\text{obj}})$ determines whether an object on the right cupboard can be pushed. It returns 1 if both poses belong to $Q_{\text{obj}}^{\text{r-cupboard}}$, and if both q_{obj} and q'_{obj} have identical orientations with respect to the global x- and y-axes.

- K_{P}: A P skill that moves a cup from the sink and places it into a cupboard while maintaining a grasp on it.
  - $K_{\text{P}}.\phi(q_{\text{obj}}, q'_{\text{obj}})$ is identical to $\phi_{\text{P}}(q_{\text{obj}}, q'_{\text{obj}})$ in the card domain, except that the predefined grasp set $\mathcal{G}_{\text{predef}}(q)$ is defined for the cup object instead of the card.

Next, we explain the goal-conditioned policy for each skill K. Each skill has its own policy π. The goal-conditioned policy of skill K, denoted $K.\pi$, consists of two distinct components: (1) the pre-contact policy $K.\pi_{\text{pre}}$, and (2) the post-contact policy $K.\pi_{\text{post}}$.

- The pre-contact policy $K.\pi_{\text{pre}}$ computes the pre-contact robot joint positions q'_r from q_{obj} and target object poses q'_{obj}.
- The post-contact policy $K.\pi_{\text{post}}$ computes a low-level action a, which includes changes of end-effector pose, proportional gain of joints, and damping of joints, to manipulate the object from current state s and desired object pose q'_{obj}.

A.4 Non-Prehensile Skill Policy

This section outlines the details for the non-prehensile (NP) skills policy, as described in Table 2. Each skill's policy $K_{\text{NP}}.\pi$ consists of π_{pre} and π_{post}. π_{post} computes low-level robot actions that manipulate the object toward the target object pose, using the state and target object pose as inputs. π_{pre} computes the robot configuration needed to execute π_{post}, with the current and target object poses as inputs.

First we describe how we train π_{post} and then explain about π_{pre}. π_{post} is trained using the Proximal Policy Optimization (PPO) algorithm, which demonstrates successful non-prehensile manipulation in Kim et al. [18]. To train the policy π_{post}, we introduce (1) the initial problem setup and (2) the

state space S, the action space A, and the reward function R used for training. The problem for training post-contact policy π_{post}, consist of (1) initial object pose $q_{\text{obj}}^{\text{init}}$, (2) initial robot configuration q_r^{init}, and (3) target object pose $q_{\text{obj}}^{\text{g}}$. Initial and target object poses are sampled from each region $K_{\text{NP}}.R$. The initial robot configuration is computed from the pre-contact policy with inputs consist of initial and target object poses $q_r^{\text{init}} = K.\pi_{\text{pre}}(q_{\text{obj}}^{\text{init}}, q_{\text{obj}}^{\text{g}})$.

For training NP skill's post-contact policy $K_{\text{NP}}.\pi_{\text{post}}$, the state space, the action space, and the reward function are defined as follows.

- **State (S):**

Extract from	Symbol	Description
Robot configuration	$q_r^{(\text{t})} \in \mathbb{R}^9$	Robot joint position
	$\dot{q}_r^{(\text{t})} \in \mathbb{R}^9$	Robot joint velocity
	$T_{\text{tool}}^{(\text{t})} \in \mathbb{R}^7$	Robot tool pose
Object configuration	$p_{\text{obj}}^{(\text{t})} \in \mathbb{R}^{24}$	Object keypoint positions
Action	$a^{(\text{t-1})} \in \mathbb{R}^{20}$	Previous action
Target object pose	$p_{\text{obj}}^{\text{g}} \in \mathbb{R}^{24}$	Target object keypoint positions

Table 4: The components of state space S of NP skill post-contact policy $K_{\text{NP}}.\pi_{\text{Post}}$

The state for the NP skill's post-contact policy, $K_{\text{NP}}.\pi_{\text{post}}$, consists of six components outlined in Table 4. The robot information includes the robot joint position $q_r^{(\text{t})}$, joint velocity $\dot{q}_r^{(\text{t})}$, and tool pose $T_{\text{tool}}^{(\text{t})}$ (The tool pose is computed from the robot joint positions $q_r^{(\text{t})}$). The previous timestep action a provides sufficient statistics [71].

The object information is represented by the object keypoint positions $p_{\text{obj}}^{(\text{t})}$ (computed from object pose $q_{\text{obj}}^{(\text{t})}$), which represents the object's geometry. From the pre-defined relative object keypoints and the object pose $q_{\text{obj}}^{(\text{t})}$, the object keypoint positions $p_{\text{obj}}^{(\text{t})}$ are computed. We pre-define eight relative object keypoints (x, y, z positions) on each object's surface. For cuboid-shaped objects like a card or a book, the eight keypoints correspond to the vertices of the cuboid. For a cup, six keypoints are located on the body, and two keypoints are positioned on the handle. These keypoint positions are used to calculate the distances between two different object poses, both in the reward function and as state components of the skill policy. The target object pose information is represented by target object keypoint positions $p_{\text{obj}}^{\text{g}}$, which are also computed from the relative object keypoints and target object pose $q_{\text{obj}}^{\text{g}}$.

- **Action (A):** The action $a \in A$ represents the control inputs applied to the robot.

Symbol	Description
$\Delta q_{\text{ee}} \in \mathbb{R}^6$	Delta end-effector pose
$k_p \in \mathbb{R}^7$	Proportional gain
$\rho \in \mathbb{R}^7$	Joint damping

Table 5: The components of action space A of NP skill post-contact policy $K_{\text{NP}}.\pi_{\text{Post}}$

The action consists of three components. The first component, Δq_{ee}, represents the delta end-effector pose, with a dimension of 6. It indicates the change in the end-effector pose, defined by the position (x, y, z) and orientation angles $(\theta_x, \theta_y, \theta_z)$. The second component, k_p, is the proportional gain for robot joints, which has a dimension of 7. It refers to the gain values for the robot's joints, excluding the two gripper joints. The third component, ρ, represents the joint damping, and it also has a dimension of 7. It specifies the damping values for the robot's joints, excluding the gripper joints.

For the robot joints (except gripper tip), the control process begins by computing the target end-effector pose using the end-effector pose $q_{\text{ee}}^{(\text{t})}$ and the change of end-effector pose Δq_{ee}. This

target pose $q_{\text{ee}}^{(t)} + \Delta q_{\text{ee}}$ is then used to solve the inverse kinematics (IK) problem, yielding the target joint positions q_r^{target}. The derivative gain k_d is calculated from the proportional gain k_p and the damping ratio ρ using the relation $k_d = \rho \cdot \sqrt{k_p}$. Finally, the PD controller computes the torque $\tau^{(t)}$ at timestep t as $\tau^{(t)} = k_p \cdot (q_r^{\text{target}} - q_r^{(t)}) - k_d \cdot \dot{q}_r^{(t)}$. The computed torque τ is applied to the joints at a frequency of 100 Hz.

For the gripper tip, no additional control is applied. The gripper width, computed by π_{pre}, is maintained.

- **Reward ($R(s_t, a_t, s_{t+1})$):**
  - **Object Keypoint Distance Reward:** Encourages moving the object closer to its target object pose. Here, $p_{\text{obj}}^{(t)}$ represents the object keypoint positions at timestep t computed from $q_{\text{obj}}^{(t)}$, $p_{\text{obj}}^{\text{g}}$ denotes the keypoint positions of the target object pose computed from $q_{\text{obj}}^{\text{g}}$, and ϵ_0^{obj}, ϵ_1^{obj} are reward hyperparameters:

$$r_{\text{obj}}^{(t)} = \frac{\epsilon_0^{\text{obj}}}{\|p_{\text{obj}}^{(t)} - p_{\text{obj}}^{\text{g}}\| + \epsilon_1^{\text{obj}}} - \frac{\epsilon_0^{\text{obj}}}{\|p_{\text{obj}}^{(t-1)} - p_{\text{obj}}^{\text{g}}\| + \epsilon_1^{\text{obj}}}$$

  - **Tip Contact Reward:** Encourages maintaining contact between the gripper tips and the object. Here, $p_{\text{tip}}^{(t)}$ represents the robot gripper tip positions at timestep t, $q_{\text{obj}}^{(t)}.\text{pos}$ represents the object position at timestep t, and $\epsilon_0^{\text{tip-obj}}$, $\epsilon_1^{\text{tip-obj}}$ are reward hyperparameters:

$$r_{\text{tip-contact}}^{(t)} = \frac{\epsilon_0^{\text{tip-obj}}}{\|p_{\text{tip}}^{(t)} - q_{\text{obj}}^{(t)}.\text{pos}\| + \epsilon_1^{\text{tip-obj}}} - \frac{\epsilon_0^{\text{tip-obj}}}{\|p_{\text{tip}}^{(t-1)} - q_{\text{obj}}^{(t-1)}.\text{pos}\| + \epsilon_1^{\text{tip-obj}}}$$

  - **Success Reward:** A success reward, r_{succ}, is given when the object is successfully manipulated to the target object pose $q_{\text{obj}}^{\text{g}}$; otherwise, the reward is 0. The distance function between two object poses, q_{obj}^1 and q_{obj}^2, is defined as $d_{SE(3)}(q_{\text{obj}}^1, q_{\text{obj}}^2) = \Delta T(q_{\text{obj}}^1, q_{\text{obj}}^2) + \alpha \Delta\theta(q_{\text{obj}}^1, q_{\text{obj}}^2)$, where ΔT represents the positional difference, $\Delta\theta$ denotes the orientational difference, and α is a weighting factor for the orientation difference, set to 0.1. The object manipulation is considered successful if the distance between the current and target object poses is less than δ_{obj}.

$$r_{\text{success}}^{(t)} = \begin{cases} r_{\text{succ}} & \text{if } d_{SE(3)}(q_{\text{obj}}^{(t)}, q_{\text{obj}}^{\text{g}}) < \delta_{\text{obj}}, \\ 0 & \text{otherwise.} \end{cases}$$

The overall reward is defined as:

$$r_{\text{NP}}^{(t)} = r_{\text{obj}}^{(t)} + r_{\text{tip-contact}}^{(t)} + r_{\text{success}}^{(t)}$$

The hyperparameters of the reward function, ϵ_0^{obj}, ϵ_1^{obj}, $\epsilon_0^{\text{tip-obj}}$, $\epsilon_1^{\text{tip-obj}}$, r_{succ}, and δ_{obj}, vary depending on the specific NP skill being trained. The values of these reward hyperparameters for each NP skill K_{NP_i} are provided in Table 6.

Domain	**Card Flip**	**Bookshelf**		**Kitchen**		
Skills	slide	topple	push	sink	l-cupboard	r-cupboard
ϵ_0^{obj}	0.02	0.3	1.0	0.2		
ϵ_1^{obj}	0.02		0.01	0.02		
$\epsilon_0^{\text{tip-obj}}$	0.0	1.0		0.3		
$\epsilon_1^{\text{tip-obj}}$	0.0	0.02	0.01	0.02		
r_{succ}	1000	500	50	500	1,000	
δ_{obj}	0.005					

Table 6: Reward hyperparameter values for training NP skill policies

For training pre-contact policy $K_{\text{NP}}.\pi_{\text{pre}}$, the state space consist of two object poses, initial object pose q_{obj}, and target object pose $q^{\text{g}}_{\text{obj}}$. The pre-contact policy outputs q'_r, which is the robot joint positions for initiating the K_{NP}. The pre-contact policy is jointly trained with π_{post} following the algorithm in Kim et al. [18].

NP skill policies utilize a multilayer perceptron (MLP) architecture to generate low-level robot actions based on the state and target object pose information. The post-contact policy π_{post} employs a five-layer MLP. The MLP has hidden dimensions of $[512, 256, 256, 128]$, input dimension 93, and an output dimension of 20. ELU is used as the activation function for the hidden layers, while Identity is applied as the final activation function. Pre-contact policy π_{pre} employs a five-layer MLP. The MLP has hidden dimensions of $[512, 256, 256, 128]$, input dimension 14, and an output dimension of 9. ELU is used as the activation function for the hidden layers, while Identity is applied as the final activation function. We summarize the hyperparamters of architecture in Table 7

	input dimensions	hidden dimensions	output dimensions	hidden activations	output activation
π_{post}	93	$[512, 256, 256, 128]$	20	ELU	Identity
π_{pre}	14		9		

Table 7: The network architecture of NP skill policies

A.5 Prehensile Skill Policy

This section describes the details of the prehensile (P) skill policy, following the structure in Table 3. Each skill's policy $K_{\text{P}}.\pi$ consists of two components: the pre-contact policy π_{pre} and the post-contact policy π_{post}.

To enable π_{pre} to generate a feasible robot configuration for grasping, we first construct a set of predefined grasp poses $\mathcal{G}_{\text{predef}}$ (mentioned in Table 3) for each object. These grasps provide candidate end-effector poses the robot can use to establish contact with the object before executing π_{post}. For cuboid objects such as the card and the book, the predefined grasp poses are generated as follows: (1) we form multiple contact points on a rectangle slightly offset from the edges on the broad surface of the object, (2) we set the grasp direction from each contact point toward the object's center of mass, and (3) we compute the grasp pose based on the contact point and the grasp direction. The grasp width is set to the thickness of the cuboid object. For the cup object, we adopt the grasp generation method from ACRONYM [72] due to the cup's asymmetric geometry. Specifically, (1) we sample random grasp poses, (2) check for collisions between the cup and the gripper, (3) close and shake the gripper, and (4) if the object remains grasped after shaking, we save the grasp pose and grasp width.

The pre-contact policy π_{pre} is implemented as a heuristic function that generates a robot configuration to establish a grasp on the object before executing π_{post}. Given the current object pose and the target object pose, π_{pre} first checks the applicability condition through $\phi(q, q')$. If $\phi(q, q') = 1$, meaning there exists a common feasible grasp between q and q', it randomly selects a grasp from the intersection $\mathcal{G}(q) \cap \mathcal{G}(q')$ and solves for a collision-free inverse kinematics (IK) solution. If multiple grasps are available, one is selected randomly. If $\phi(q, q') = 0$, π_{pre} instead randomly selects a grasp from the feasible grasp set $\mathcal{G}(q)$ associated with the initial object pose.

The post-contact policy π_{post} computes low-level robot actions to manipulate the object from its current pose toward the desired pose. It takes as input the robot's state and the target object pose. The policy π_{post} is trained using the Proximal Policy Optimization (PPO) algorithm. The training problem for π_{post} consists of (1) an initial object pose $q^{\text{init}}_{\text{obj}}$, (2) an initial robot configuration q^{init}_r, and (3) a target object pose $q^{\text{g}}_{\text{obj}}$. The initial and goal object poses are sampled from the valid region Q_{obj}, and the initial robot configuration q^{init}_r is computed using the pre-contact policy as $q^{\text{init}}_r = \pi_{\text{pre}}(q^{\text{init}}_{\text{obj}}, q^{\text{g}}_{\text{obj}})$.

For training the post-contact policy π_{post}, the state space, the action space, and the reward function are defined as follows.

- **State (S):**

Extract from	Symbol	Description
Robot configuration	$q_r^{(t)} \in \mathbb{R}^9$	Robot joint position
	$\dot{q}_r^{(t)} \in \mathbb{R}^9$	Robot joint velocity
	$T_{\text{tool}}^{(t)} \in \mathbb{R}^7$	Robot tool pose
	$p_{\text{ee}}^{(t)} \in \mathbb{R}^{24}$	Robot end-effector keypoint positions
	$p_{\text{tip}}^{(t)} \in \mathbb{R}^6$	Robot gripper tip positions
Object configuration	$p_{\text{obj}}^{(t)} \in \mathbb{R}^{24}$	Object keypoint positions
Action	$a^{(t-1)} \in \mathbb{R}^{20}$	Previous action
Target object pose	$p_{\text{obj}}^{g} \in \mathbb{R}^{24}$	Target object keypoint positions

Table 8: The components of state space S of P skill policy π_{P}

The state space consists of eight components outlined in Table 8. The end-effector pose is computed from the robot joint positions $q_r^{(t)}$. Eight keypoints are defined at the vertices of the end-effector. Using these keypoints and the end-effector pose, the end-effector keypoint positions $p_{\text{ee}}^{(t)}$ are computed. Similarly, the gripper tip positions $p_{\text{tip}}^{(t)}$ are computed from the robot joint positions $q_r^{(t)}$. The remaining state components are identical to the non-prehensile skill's post-contact policy's state components.

- **Action (A):** The action space of the prehensile skill is identical to that of the NP skill. For the gripper tip, control is applied with a width of 0 to maintain the grasp on the object.

- **Reward ($R(s_t, a_t, s_{t+1})$):**
  - **Object Keypoint Distance Reward:** The reward $r_{\text{obj}}^{(t)}$ computation is identical to the object keypoint distance reward used in non-prehensile post-contact policy training.
  - **Object Rotation Reward:** Encourages aligning the object orientation with its target object orientation. Here, $q_{\text{obj}}^{(t)}.\theta$ represents the object pose orientation at timestep t, and $q_{\text{obj}}^{g}.\theta$ denotes the target object pose orientation:

$$r_{\text{rot}}^{(t)} = \frac{\epsilon_0^{\text{rot}}}{\|q_{\text{obj}}^{(t)}.\theta - q_{\text{obj}}^{g}.\theta\| + \epsilon_1^{\text{rot}}} - \frac{\epsilon_0^{\text{rot}}}{\|q_{\text{obj}}^{(t-1)}.\theta - q_{\text{obj}}^{g}.\theta\| + \epsilon_1^{\text{rot}}}$$

  - **Relative Grasp Reward:** Ensures the gripper tips maintain a consistent grasp on the object. Here, $p_{\text{ee-rel}}^{(t)}$ represents the relative position of the robot end-effector keypoints. It is computed from the absolute end-effector keypoints $p_{\text{ee}}^{(t)}$ with respect to the object pose $q_{\text{obj}}^{(t)}$ at timestep t:

$$r_{\text{grasp}}^{(t)} = w^{\text{grasp}} \|p_{\text{ee-rel}}^{(t)} - p_{\text{ee-rel}}^{(t-1)}\|$$

  - **Success Reward:** The reward $r_{\text{success}}^{(t)}$ computation is identical to the success reward used in non-prehensile post-contact policy training.

The overall reward is defined as:

$$r_{\text{P}} = r_{\text{obj}}^{(t)} + r_{\text{rot}}^{(t)} + r_{\text{grasp}}^{(t)} + r_{\text{success}}^{(t)}$$

The hyperparameters of the reward function, $\epsilon_0^{\text{object}}$, $\epsilon_1^{\text{object}}$, $\epsilon_0^{\text{rotation}}$, $\epsilon_1^{\text{rotation}}$, w^{grasp}, r_{succ}, and δ_{obj}, vary depending on the specific P skill being trained. The values of these reward hyperparameters for each P skill K_{P} are provided in Table 9.

Domain	Card Flip	Bookshelf	Kitchen
$\epsilon_0^{\text{object}}$	0.2	0.15	1
$\epsilon_1^{\text{object}}$	0.02		0.015
$\epsilon_0^{\text{rotation}}$	0.0	0.15	0.0
$\epsilon_1^{\text{rotation}}$	0.0	0.1	0.0
w^{grasp}	0.0	10.0	15.0
r_{succ}	1000		
δ_{obj}	0.005		

Table 9: Reward hyperparameter values for training P skill post-contact policy

We train the P skill policy with Proximal Policy Optimization (PPO) [32].

P skill policies utilize a multilayer perceptron (MLP) architecture to generate low-level robot actions based on the state and target object pose information. $K_{\text{P}}.\pi_{\text{P}}$ employs a five-layer MLP with an input dimension of 123. Other components of the network architecture are identical to the NP skill post-contact policy's network architecture. The hyperparameters of network is summarized in Table 10

	input dimensions	hidden dimensions	output dimensions	hidden activations	output activation
π_{post}	123	$[512, 256, 256, 128]$	20	ELU	Identity

Table 10: The network architecture of P skill post-contact policy

B Detail of SPIN

In this section, we describe (1) additional algorithms used in Skill-RRT, (2) the training procedure of the connector policy, (3) the state-action structure and network architecture used for imitation learning in SPIN, and (4) the domain randomization ranges in simulation, along with the level of noise added to the observations for real-world deployment.

B.1 Detail of Skill-RRT

This section provides the algorithms used in Skill-RRT. These include ComputeConnectingNode, UnifSmplSkillAndSubgoal, GetApplicableNearestNode, and Failed. Then, we describe Skill-RRT-Batch, an extended Skill-RRT algorithm that extends nodes in parallel to accelerate planning and data collection times using GPU-based simulation, specifically Isaac Gym [47]. Note that we use Skill-RRT-Batch instead of the vanilla Skill-RRT throughout our experiments.

Across the algorithms, the distance function $d_{SE(3)}$, which is used to determine the success of NP and P skills, is used in the same way.

Algorithm 3 provides the pseudocode for the UnifSmplSkillAndSubgoal function. This function takes as input the set of skills $\mathcal{K}$, and the object regions Q_{obj}. It outputs a uniformly sampled skill K and the desired object pose for the skill, q_{obj}. The algorithm begins by uniformly sampling a skill K from the set of skills $\mathcal{K}$ (L1). Then, the desired object pose q_{obj} is set to the goal object pose q^g_{obj} with a goal sampling probability p_g (L2–4), or it is uniformly sampled from the object regions Q_{obj} otherwise (L5–6). Finally, the algorithm returns the sampled skill K and the desired object pose for the skill q_{obj}.

Algorithm 3 UnifSmplSkillAndSubgoal($\mathcal{K}, Q_{\text{obj}}$)

1: $K \leftarrow$ UniformSample($\mathcal{K}$)
2: $p \leftarrow$ rand(1)
3: **if** $p < p_g$ **then**
4: $\quad q_{\text{obj}} \leftarrow q^g_{\text{obj}}$
5: **else**
6: $\quad q_{\text{obj}} \leftarrow$ UniformSamplePose(Q_{obj})
7: **return** K, q_{obj}

Algorithm 4 provides the pseudocode for the GetApplicableNearestNode function. It takes in the tree T, the skill K for which feasibility is to be checked, and the desired object pose for the skill, q_{obj}. If the skill K is a non-prehensile skill, we compute all feasible nodes for the given skill and desired object pose in the tree using the feasibility checker $K.\phi(s, q_{\text{obj}})$. We then find the nearest node, v_{near}, to the desired object pose q_{obj} among the feasible nodes using the distance function $d_{SE(3)}$, and return v_{near} (L1-3). If the skill K is a prehensile skill, we first compute the closest node, v_{close}, to the desired object pose q_{obj} in the tree (L5). If the closest node v_{close} passes the feasibility check using the feasibility checker for the prehensile skill, v_{near} is set to v_{close} (L6-7). Otherwise, v_{near} is set to the empty set (L8-9). The different order of finding the nearest node and performing the feasibility check for prehensile skills is due to the computational expense of the feasibility checker of the prehensile skill, as it involves collision checking.

Algorithm 4 `GetApplicableNearestNode`(T, K, q_{obj})

if $K \in \{K_{\text{NP}_1}, \ldots, K_{\text{NP}_n}\}$ **then**
  $V_{\text{feasible}} = \{v \mid v \in T.V, K.\phi(v.s, q_{\text{obj}}) == 1\}$
  $v_{\text{near}} \leftarrow \arg\min_{v \in V_{\text{feasible}}} d_{SE(3)}(v.q_{\text{obj}}, q_{\text{obj}})$
else ▷ If prehensile skill
  $v_{\text{close}} \leftarrow \arg\min_{v \in T.V} d_{SE(3)}(v.q_{\text{obj}}, q_{\text{obj}})$
  if $K.\phi(v_{\text{close}}.s, q_{\text{obj}})$ **then**
    $v_{\text{near}} \leftarrow v_{\text{close}}$
  else
    $v_{\text{near}} \leftarrow \emptyset$
return v_{near}

Algorithm 5 shows the `ComputeConnectingNode` function. The function takes as input the target robot joint configuration q'_r, the skill K to be simulated, the connector set $\mathcal{C}$ which includes connector policies, and the node v from which the connector policy or teleportation is applied. The algorithm begins by checking if $\mathcal{C}$ is empty, in which case this will instantiate `Lazy Skill-RRT`(L1). In this case, we create a connecting node v_{connect} with empty skill and object pose, with the same state as v, except the robot joint configuration is teleported to q'_{r} (L2-4). Otherwise, we retrieve the connector π_C for the given skill K (L6), simulate it from $v.s$ with pre-contact configuration q'_{r} as a goal (L7), and create v_{connect} using π_C, q'_{r}, and the resulting state s' (L8).

Algorithm 5 `ComputeConnectingNode`$(q'_{\text{r}}, K, \mathcal{C}, v)$

1: IsLazy $\leftarrow \mathcal{C} == \emptyset$
2: **if** IsLazy **then**
3: $v_{\text{connect}} \leftarrow (\emptyset, \emptyset, v.s)$
4: $v_{\text{connect}}.s.q_{\text{r}} \leftarrow q'_{\text{r}}$ ▷ Teleport the q_r to q'_r
5: **else**
6: $\pi_C \leftarrow$ `GetConnectorForSkill`$(\mathcal{C}, K)$
7: $s' \leftarrow f_{\text{sim}}(v.s, \pi_C(v.s; q'_{\text{r}}))$
8: $v_{\text{connect}} \leftarrow (\pi_C, q'_{\text{r}}, s')$
return v_{connect}

Algorithm 6 describes the `ComputePreSkillConfig` function. This function takes as input a skill K, the current object pose $v.s.q_{\text{obj}}$, and the target object pose q'_{obj}. The goal is to compute a pre-contact robot joint configuration that positions the robot appropriately to execute the skill K starting from $v.s.q_{\text{obj}}$ and targeting q'_{obj}. The algorithm first retrieves the pre-contact policy $K.\pi_{\text{pre}}$ associated with the skill K (L1). It then applies this policy to the object pose transition $(v.s.q_{\text{obj}}, q'_{\text{obj}})$ to obtain the corresponding robot configuration q'_{r} (L2), which serves as the starting point for the skill execution. The computed configuration q'_{r} is returned as the output.

Algorithm 6 `ComputePreSkillConfig`$(K, v.s.q_{\text{obj}}, q'_{\text{obj}})$

1: $K.\pi_{\text{pre}} \leftarrow$ `GetPreContactPolicy`(K)
2: $q'_{\text{r}} \leftarrow K.\pi_{\text{pre}}(v.s.q_{\text{obj}}, q'_{\text{obj}})$
 return q'_{r}

Algorithm 7 checks whether a skill fails given the state s and the desired object pose q_{obj}. If the distance between the object pose in the state $s.q_{\text{obj}}$ and the desired object pose q_{obj} is less than the predefined threshold δ_{obj}, the algorithm returns false. Otherwise, it returns true.

Algorithm 7 `Failed`(s, q_{obj})

if $d_{SE(3)}(s.q_{\text{obj}}, q_{\text{obj}}) < \delta_{\text{obj}}$ **then**
  $fail \leftarrow False$
else
  $fail \leftarrow True$
return $fail$

We now describe `Skill-RRT-Batch` in Algorithm 8, a parallelized version of the original `Skill-RRT`. The inputs and outputs are the same as those of the original `Skill-RRT`. The main differences in `Skill-RRT-Batch` are: 1) uniformly sampling skills and desired object poses in batches using the `UnifSmplSkillAndSubgoalBatch` function (L5); 2) finding the feasible and nearest node $\mathbf{v}_{\text{near}}$ using the `GetApplicableNearestNodeBatch` function (L6); and 3) extending nodes in parallel with GPU-based simulation via the `ExtendBatch` function. Bold letters indicate a batch of components. We do not rewrite the batched versions of the functions because single components (normal letters) are simply replaced by batches of components (bold letters).

Algorithm 8 `Skill-RRT-Batch`$(s_0, q^g_{\text{obj}}, \mathcal{K}, \mathcal{C}, Q_{\text{obj}})$

1: $T = \emptyset$
2: $v_0 \leftarrow \{(\emptyset, \emptyset), s_0\}$
3: T.`AddNode`$(parent = \emptyset, child = v_0)$
4: **for** $i = 1$ **to** N_{max} **do**
5: $\mathbf{K}, \mathbf{q}_{\text{obj}} \leftarrow$`UnifSmplSkillAndSubgoalBatch`$(\mathcal{K}, Q_{\text{obj}})$
6: $\mathbf{v}_{\text{near}} \leftarrow$ `GetApplicableNearestNodeBatch`$(T, \mathbf{K}, \mathbf{q}_{\text{obj}})$
7: **if** $\mathbf{v}_{\text{near}}$ is $\emptyset$ **then**
8: **continue**
9: `ExtendBatch`$\big(T, \mathbf{v}_{\text{near}}, \mathbf{K}, \mathbf{q}_{\text{obj}}, \mathcal{C}\big)$
10: **if** `Near`(q^g_{obj}, T) **then**
11: **Return** `Retrace`(q^g_{obj}, T)
12: **Return** None

B.2 Connector Policy Training

To train the connector policy π_C, we introduce the state space S, the action space A, and the reward function R used for training. The problem setup for connector policy is provided in Section 3.2.

For training connector policy π_C, the state space, the action space, and the reward function are defined as follows.

- **State (S):**

Extract from	Symbol	Description
Robot configuration	$q_r^{(\text{t})} \in \mathbb{R}^9$	robot joint position
	$\dot{q}_r^{(\text{t})} \in \mathbb{R}^9$	Robot joint velocity
	$T_{\text{tool}}^{(\text{t})} \in \mathbb{R}^7$	Robot tool pose
	$p_{\text{ee}}^{(\text{t})} \in \mathbb{R}^{24}$	Robot end-effector keypoint positions
	$p_{\text{tip}}^{(\text{t})} \in \mathbb{R}^6$	Robot gripper tip positions
Object configuration	$p_{\text{obj}}^{(\text{t})} \in \mathbb{R}^{24}$	Object keypoint positions
Action	$a^{(\text{t-1})} \in \mathbb{R}^{21}$	Previous action
Simulator	$\mathbb{1}^{(\text{t})}_{\text{gripper-execute}}$	Gripper action executability
Target robot configuration	$p_{\text{ee}}^{\text{g}} \in \mathbb{R}^{24}$	Target end-effector keypoint positions
	$p_{\text{tip}}^{\text{g}} \in \mathbb{R}^{24}$	Target gripper tip positions

Table 11: The components of state space S of connector policy π_C

The state for the skill's connector policy, $K.\pi_C$, consists of ten components outlined in Table 11. $\mathbb{1}^{(t)}_{\text{gripper-execute}}$, provided by the simulator, represents a binary indicator of the executability of the gripper action, where a value of 1 indicates that the action is executable. The gripper action is executable every 1.2 seconds because the Franka Research 3 gripper cannot accept new commands until the previous gripper command is completed. The goal is represented by the target end-effector keypoint positions p^{g}_{ee} and the target gripper tip positions p^{g}_{tip}, which define the target configurations for the robot to achieve. The remaining state components are identical to the state components of the prehensile skill post-contact policy.

- **Action (A):** The action $a \in A$ represents the control inputs applied to the robot. The action consists of four components as show in Table 12. The delta end-effector pose, the proportional gain for robot joints, and joint damping correspond to the action components of the NP skill's post-contact policy. The fourth component, q_{width}, is the target gripper tip width. For the robot joints (excluding the gripper tip), the control process follows the same procedure as the post-contact policy control process used in the non-prehensile skill. For the gripper tip, the gripper width is adjusted to q_{width}.

Symbol	Description
$\Delta q_{\text{ee}} \in \mathbb{R}^6$	Delta end-effector pose
$k_p \in \mathbb{R}^7$	Proportional gain
$\rho \in \mathbb{R}^7$	Joint damping
$q_{\text{width}} \in \mathbb{R}$	Target gripper width

Table 12: The components of action space A of connector policy π_C

- **Reward ($R(s_t, a_t, s_{t+1})$):**
  1. **End-Effector Distance Reward:** Encourages the end-effector to move closer to its target position. Here, $p^{(t)}_{\text{ee}}$ represents the end-effector keypoint positions at timestep t, and p^{g}_{ee} denotes the target end-effector keypoint positions:

$$r^{(t)}_{\text{ee}} = \epsilon^{\text{ee}}_0 \big(\exp(-\epsilon^{\text{ee}}_1 \|p^{(t)}_{\text{ee}} - p^{g}_{\text{ee}}\|_2) - \exp(-\epsilon^{\text{ee}}_1 \|p^{(t\text{-}1)}_{\text{ee}} - p^{g}_{\text{ee}}\|_2) \big)$$

  2. **Gripper Tip Position Reward:** Aligns the gripper tips with their target positions. Here, $p^{(t)}_{\text{ee}}$ represents the gripper tip positions at timestep t, and p^{g}_{tip} denotes the target tip positions:

$$r^{(t)}_{\text{tip}} = \epsilon^{\text{tip}}_0 \big(\exp(-\epsilon^{\text{tip}}_1 \|p^{(t)}_{\text{ee}} - p^{g}_{\text{tip}}\|_2) - \exp(-\epsilon^{\text{tip}}_1 \|p^{(t\text{-}1)}_{\text{ee}} - p^{g}_{\text{tip}}\|_2) \big)$$

  3. **Object Movement Penalty:** Penalizes unnecessary object movement to ensure the preconditions of subsequent skills remain valid. Here, $p^{(t)}_{\text{obj}}$ represents the object keypoint positions at timesteps t:

$$r^{(t)}_{\text{obj-move}} = -w^{\text{move}} \|p^{(t)}_{\text{obj}} - p^{(t\text{-}1)}_{\text{obj}}\|_2$$

  4. **Success Reward:** A success reward, r_{succ}, is given when both the end-effector and gripper tip reach their respective target positions, p^{g}_{ee} and p^{g}_{tip}, within the allowable error thresholds δ_{ee} and δ_{tip}.

$$r^{(t)}_{\text{success}} = \begin{cases} r_{\text{succ}} & \text{if } ||p^{(t)}_{\text{ee}} - p^{g}_{\text{ee}}||_2 < \delta_{\text{ee}}, \text{ and } ||p^{(t)}_{\text{tip}} - p^{g}_{\text{tip}}||_2 < \delta_{\text{tip}}, \\ 0 & \text{otherwise} \end{cases}$$

The overall reward is defined as:

$$r^{(t)}_{\text{connector}} = r^{(t)}_{\text{ee}} + r^{(t)}_{\text{tip}} + r^{(t)}_{\text{obj-move}} + r^{(t)}_{\text{success}}$$

The hyperparameters of the reward function, ϵ_0, ϵ_1, ω, ϵ_{vel}, δ_{ee}, and δ_{tip}, vary depending on the specific connector being trained. The values of these reward hyperparameters for each connector are provided in Table 13.

Domain	**Card Flip**		**Bookshelf**			**Kitchen**			
Skills	slide	prehensile	topple	prehensile	push	sink	prehensile	l-cupboard	r-cupboard
ϵ_0^{ee}	40								
ϵ_1^{ee}	0.9					1.0			
ϵ_0^{tip}	40							30	
ϵ_1^{tip}	1.0								
ω^{move}	-0.3						-3.0	-10	
r_{succ}	1000					150		100	
δ_{ee}	0.01								
δ_{tip}	0.003								

Table 13: Reward hyperparameter values for training connector policy

Connector policies utilize a multilayer perceptron (MLP) architecture to generate low-level robot actions based on the state and target robot configuration. Each connector policy π_C employs a five-layer MLP with an input dimension of 131 and an output dimension of 21. Other components of the network architecture are identical to those of the NP skill post-contact policy's network architecture.

	input dimensions	hidden dimensions	output dimensions	hidden activations	output activation
π_C	131	$[512, 256, 256, 128]$	21	ELU	Identity

Table 14: The network architecture of the connector policy

B.3 Imitation Learning Detail

Symbol	**Description**
$q_r^{(\text{t})} \in \mathbb{R}^9$	Robot joint position
$q_r^{(\text{t-1})} \in \mathbb{R}^9$	Previous robot joint position
$p_{\text{ee}}^{(\text{t})} \in \mathbb{R}^{24}$	Robot end-effector keypoint positions (computed from $q_r^{(\text{t})}$)
$p_{\text{tip}}^{(\text{t})} \in \mathbb{R}^6$	Robot gripper tip positions (computed from $q_r^{(\text{t})}$)
$p_{\text{ee-rel}}^{(\text{t})} \in \mathbb{R}^{24}$	Relative robot end-effector keypoint positions w.r.t $q_{\text{obj}}^{(\text{t})}$
$p_{\text{tip-rel}}^{(\text{t})} \in \mathbb{R}^6$	Relative robot gripper tip positions w.r.t $q_{\text{obj}}^{(\text{t})}$
$p_{\text{obj}}^{(\text{t})} \in \mathbb{R}^{24}$	Object keypoint positions (computed from $q_{\text{obj}}^{(\text{t})}$)
$\mathbb{1}_{\text{gripper-execute}}^{(\text{t})}$	Gripper action executability (executable every 1.2 seconds)
$a_{\text{width}}^{(\text{t-1})} \in \mathbb{R}$	Previous timestep's robot gripper width action
$p_{\text{obj}}^{\text{g}} \in \mathbb{R}^{24}$	Goal object keypoint positions

Table 15: The components of state space S of distillation policy.

In this section, we describe the diffusion policy used for imitation learning, including its architecture, input/output structure, and training configuration.

We train the diffusion policy with a U-Net backbone from the diffusion policy [5] codebase. To achieve faster inference times, we remove action chunking and state history.

For the state components, as shown in Table 15, we use the previous robot joint position $q_r^{(\text{t-1})}$ instead of the robot joint velocity $\dot{q}_r^{(\text{t})}$ due to the large sim-to-real gap in joint velocity. The relative positions of the robot's end-effector keypoints $p_{\text{ee-rel}}^{(\text{t})}$ and gripper tip $p_{\text{tip-rel}}^{(\text{t})}$, with respect to the object pose q_{obj},

Symbol	Description
$\Delta q_{\text{joint}} \in \mathbb{R}^7$	Changes in the robot joint positions (except the two gripper joints)
$q_{\text{width}} \in \mathbb{R}$	Target gripper width
$k_p \in \mathbb{R}^7$	Proportional gain of robot joints (except the two gripper joints).
$\rho \in \mathbb{R}^7$	Damping of robot joints (except the two gripper joints).

Table 16: The components of action space A of distillation policy.

are used to explicitly represent the relationship between the object and the robot. We also use $a_{\text{width}}^{\text{(t-1)}}$ to identify the previous width action in order to provide previous gripper commands for efficient execution. Other state components ($q_r^{\text{(t)}}, p_{\text{ee}}^{\text{(t)}}, p_{\text{tip}}^{\text{(t)}}, p_{\text{obj}}^{\text{(t)}}$, and $p_{\text{obj}}^{\text{g}}$) correspond to the same components in the state components of the prehensile skill post-contact policy. The state component $\mathbb{1}_{\text{gripper-execute}}^{\text{(t)}}$ corresponds to the connector policy's state component.

For the action components, as described in Table 16, it consists of four components to perform torque control for non-prehensile manipulation. Their gains and damping values correspond to the same components in the action components of the non-prehensile skill post-contact policy. The distillation policy uses changes in the robot joint positions Δq_{joint} instead of changes in the end-effector Δq_{ee}, eliminating the need for inverse kinematics (IK) computation and enabling faster inference. Hyperparameters related to training diffusion policies are summarized in Table 17. These parameters remain consistent across the card flip, bookshelf, and kitchen domains.

To	Ta	Tp	Hidden Dimension	LR	Weight Decay	Batch Size	D-Iters Train	D-Iters Eval	D-Iters Emb Dim
1	1	1	[256,512,1024,2048]	1e-4	1e-6	4096	100	8	256

Table 17: Hyperparameters of U-Net based diffusion policy. To: observation horizon, Ta: action horizon, Tp: action prediction horizon, Hidden Dimension: hidden dimension of U-Net, LR: learning rate, Weight Decay: weight decay of optimizer, Batch Size: size of mini batch in training, D-Iters Train: number of training diffusion iterations, D-Iters Eval: number of inference DDIM iterations, D-Iters Emb Dim: embedding size of diffusion timestep

B.4 Domain Randomization

For sim-to-real transfer, we apply domain randomization throughout the entire pipeline, including skill training, `Skill-RRT`, and data collection. To mimic perception noise, we inject noise into the object pose, robot joint positions, and end-effector pose. We apply different scales of noise to the object pose, as the perception noise of the object pose increases depending on the environment. For example, when a book is fitted between two other books in the bookshelf domain, the noise in the object pose is larger because the book is only partially visible. We also randomize the friction of the environment since real-world friction is unknown and add random noise to the commanded torque to reflect real-world noise effects in robot controller. The noises for each domain are summarized in Table 18.

	Range		
Domain	**Card Flip**	**Bookshelf**	**Kitchen**
Object Pose (Position)	$+N[0.0, 0.003]$	$+N[0.0, 0.005]$	$+N[0.0, 0.003]$
Object Pose (Orientation)	$+N[0.0, 0.03]$	$+N[0.0, 0.05]$	$+N[0.0, 0.03]$
Robot joint position	$+N[0.0, 0.005]$		
EE Pose (Position)	$+N[0.0, 0.001]$		
EE Pose (Orientation)	$+N[0.0, 0.01]$		
Environment friction	$\times U[0.8, 1.2]$		
Robot EE surface friction	$\times U[0.9, 1.1]$		
Object mass	$\times U[0.8, 1.2]$		
Torque noise	$+N[0.0, 0.03]$		

Table 18: Ranges of domain randomization. $U[\text{min}, \text{max}]$ denotes uniform distribution, and $N[\mu, \sigma]$ denotes normal distribution. The symbol "+" represents the summation operation, and "×" represents the product operation.

C Detail of Main Experiments

In this section, we present (1) the detailed training setup for the baselines used in our main experiment (Section 4.1 Table 1), (2) an in-depth analysis of the main experimental results, and (3) the setup for the real-world experiments in Section 4.2.

C.1 Detail of Baselines

C.1.1 Baseline PPO Detail

Baseline PPO is a flat reinforcement learning method whose training pipeline follows the same process as skill training. To train the policy π_{post}, we detail: (1) the initial problem setup, (2) the state space S, the action space A, and the reward function R, and (3) a policy architecture used for training.

The problem for training the post-contact policy, π_{post}, consists of: (1) the initial object pose $q_{\text{obj}}^{\text{init}}$, (2) the initial robot configuration q_r^{init}, and (3) the target object pose $q_{\text{obj}}^{\text{g}}$. We randomly sample the initial and desired object poses, the same as in Skill-RRT in Appendix A.1. Subsequently, we compute q_r^{init} by $\pi_{\text{pre}}(q_{\text{obj}}^{\text{init}}, q_{\text{obj}}^{\text{g}})$. If the computed q_r^{init} does not cause collisions, either between the robot and the environment or between the robot and the object, the problem is generated. Otherwise, it is excluded from the dataset.

- **State (S):** The components of the state space for the PPO policy are identical to those of the distillation policy. However, the PPO policy uses the robot's joint velocity $\dot{q}_r^{\text{(t)}}$ instead of the previous joint position $q_r^{\text{(t-1)}}$ and excludes the gripper action executability $\mathbb{1}_{\text{gripper-execute}}$. Additionally, it incorporates not only the previous gripper width action $a_{\text{width}}^{\text{(t-1)}}$ but also all previous actions $a^{\text{(t-1)}}$ as components of the state space.

- **Action (A):** The action components are identical to action components of connector policy.

- **Reward (R):** The reward function $R(s_t, a_t, s_{t+1})$ is designed to encourage the robot to manipulate object to the goal object pose $q_{\text{obj}}^{\text{g}}$. The reward consists of three main components:

  - **Object Keypoint Distance Reward:** The reward $r_{\text{obj}}^{\text{(t)}}$ computation is identical to the object keypoint distance reward used in non-prehensile skill post-contact policy training.
  - **Tip Contact Reward:** The reward $r_{\text{tip-contact}}^{\text{(t)}}$ computation is identical to the tip contact reward used in non-prehensile post-contact policy training.
  - **Domain-Oriented Reward:** Encourages the successful completion of domain-specific objectives. The exact reward varies depending on the domain as shown in Table 19:

$$r_{\text{domain}}^{\text{(t)}} = \epsilon_0^{\text{domain}} \cdot \mathbb{I}[\text{domain-specific conditions}]$$

Domain	domain-specific conditions
Card Flip	The card is flipped
Bookshelf	The book is placed on the box
Kitchen	The cup is placed on the shelf

Table 19: domain-specific conditions for training the Baseline PPO policy

  - **Success Reward:** The reward $r_{\text{success}}^{\text{(t)}}$ computation is identical to the success reward used in non-prehensile post-contact policy training.

The overall reward is defined as:

$$r_{\text{PPO}}^{\text{(t)}} = r_{\text{obj}}^{\text{(t)}} + r_{\text{tip-obj}}^{\text{(t)}} + r_{\text{domain}}^{\text{(t)}} + r_{\text{success}}^{\text{(t)}}$$

The hyperparameters of the reward function, ϵ_0^{obj}, ϵ_1^{obj}, $\epsilon_0^{\text{tip-obj}}$, $\epsilon_1^{\text{tip-obj}}$, $\epsilon_0^{\text{domain}}$ and r_{succ}, are provided in Table 20.

Domain	Card Flip	Bookshelf	Kitchen
ϵ_0^{obj}	0.02		
ϵ_1^{obj}	0.02		
$\epsilon_0^{\text{tip-obj}}$	0.03		
$\epsilon_1^{\text{tip-obj}}$	0.03		
$\epsilon_0^{\text{domain}}$	0.5		
r_{succ}	1000		
δ_{obj}	0.005		

Table 20: Reward hyperparameter values for training the Baseline PPO policy

The baseline PPO policies utilize a multilayer perceptron (MLP) architecture to generate low-level robot actions based on the state and goal object pose information. Each policy, π_{pre} and π_{post}, employs a five-layer MLP. π_{pre} and π_{post} have input dimensions of 14 and 147, respectively. π_{pre} and π_{post} have output dimensions of 9 and 21, respectively. The other components of the architecture are identical to those of the non-prehensile skill post-contact policy's network architecture.

	input dimensions	hidden dimensions	output dimensions	hidden activations	output activation
π_{pre}	14	$[512, 256, 256, 128]$	9	ELU	Identity
π_{post}	147		21		

Table 21: The network architecture of the Baseline PPO policies

C.1.2 Baseline `MAPLE` Detail

Baseline `MAPLE` is a hierarchical reinforcement learning method that includes the task policy π_{tsk} and the parameter policy $\pi_{\text{parameter}}$. The algorithm is implemented based on Soft Actor-Critic [73], consistent with the original implementation. However, we modify the `MAPLE` implementation to incorporate a parallelized environment, Isaac Gym, and to adapt `MAPLE` to our PNP problem. First, we remove the affordance reward, which in `MAPLE` guides the high-level policy toward the desired manipulation region. Instead, we integrate the connectors and skills, incorporating an applicability checker ϕ. The connector, which moves the robot to a state where the corresponding skill is applicable, is executed only when the predicted desired object pose is applicable (i.e., when the corresponding ϕ holds true); otherwise, it is not executed. This replaces the need for an affordance reward. Furthermore, we replace the explicit initial end-effector position parameter, x_{reach}, which serves as an input to skills in `MAPLE`, with the output of the pre-contact policy π_{pre} of each skill. Therefore, the initial robot position is computed by π_{pre}, and the connector policy moves the robot to this computed position before skill execution. Additionally, we eliminate atomic primitives, low-level actions used to fill in gaps that cannot be fulfilled by skills, since our connectors are already trained to handle these gaps.

To train the policies, we outline: (1) the initial problem setup, (2) the state space S, the action space A, and the reward function R and, (3) hyperparameters used for training.

The problem for training the policies, π_{tsk} and $\pi_{\text{parameter}}$, consists of: (1) the initial object pose $q_{\text{obj}}^{\text{init}}$, and (2) the target object pose $q_{\text{obj}}^{\text{g}}$. We randomly sample the initial and target object poses, the same as in Skill-RRT in Appendix A.1.

- **State (S):** The components of the state space for the `MAPLE` policies are identical to those of the distillation policy. However, the `MAPLE` policies utilize the robot's joint velocity $\dot{q}_r^{(\text{t})}$ instead of the previous joint position $q_r^{(\text{t-1})}$ and omits both the gripper action executability $\mathbb{1}_{\text{gripper-execute}}$ and previous timestep's robot gripper width action $a_{\text{width}}^{(\text{t-1})}$.

- **Action (A):** The action $(K, \hat{q}_{\text{sg}}^{\text{obj}}) \in A$ represents skills K and its corresponding normalized desired object pose $\hat{q}_{\text{sg}}^{\text{obj}}$ for the control module $\pi_{\text{skill}}(\cdot; q_{\text{sg}}^{\text{obj}})$. The normalized desired object pose is unnormalized to align with the region of the corresponding skill.

- **Reward (R):** The reward function $R(s, K, \hat{q}_{\text{sg}}^{\text{obj}}, s')$ is designed as a sparse reward to encourage the robot to manipulate the object to the goal object pose $q_{\text{g}}^{\text{obj}}$, as defining a dense reward for this task is challenging. The reward consists of :

  – **Feasible Skill Reward:** Encourages the policy to choose feasible skills.

  $$r_{\text{feasible}}^{(t)} = \epsilon_0^{\text{feasible}} \cdot \mathbb{1}[K.\phi(s, q_{\text{sg}}^{\text{obj}})]$$

  – **Success Reward:** The reward $r_{\text{success}}^{(t)}$ computation is identical to the success reward used in non-prehensile post-contact policy training.

  The overall reward is defined as:

  $$r_{\texttt{MAPLE}}^{(t)} = r_{\text{feasible}}^{(t)} + r_{\text{success}}^{(t)}$$

  The hyperparameters of the reward function, $\epsilon_0^{\text{feasible}} = 0.1$, $\delta_{\text{obj}} = 0.005$ and $r_{\text{succ}} = 100$, are the same for all tasks.

Hyperparameter	Value
Hidden sizes (all networks)	1024, 512, 256, 256
Q network and policy activation	ReLU
Q network output activation	None
Policy network output activation	tanh
Optimizer	Adam
Batch Size	4096
Learning rate (all networks)	3e-5
Target network update rate τ	5e-4
# Training steps per epoch	1000
Replay buffer size	1e6
Discount factor	0.99
Reward scale	100.0
Automatic entropy tuning	True
Target Task Policy Entropy	$0.50 \times \log(k)$, k is number of skills
Target Parameter Policy Entropy	$-\max_a d_a$

Table 22: Hyperparameters for the baseline MAPLE

Hyperparameters related to training MAPLE are summarized in Table 22. These parameters remain consistent across all the tasks.

Additional Analysis MAPLE struggles in Card Flip and Kitchen domains due to the narrow passage problem [25]. For instance, in the Card Flip domain, the card can only be flipped near the edges of the table, which constitute a very small region in Q_{obj}. Moreover, even at the edge, to prevent the card from dropping during the transition from sliding to the prehensile skill, the card must be positioned precisely: it must be graspable while remaining sufficiently close to the table surface to avoid falling during the transition Similarly, in the Kitchen domain, the robot must position the cup such that the prehensile skill can be applied following the non-prehensile skill in the sink. The set of cup poses suitable for applying the prehensile skill is very narrow: the pose must be collision-free, and a valid inverse kinematics solution must exist. As a result, most attempts at exploring object subgoal poses lead to failure for MAPLE, which ends up learning only locally optimal behaviors, such as bringing the card to the edge of the table but avoiding the prehensile action to prevent the card from falling. In the Bookshelf domain, where the robot simply needs to topple the book to enable grasping, MAPLE achieves an 83% success rate.

C.1.3 Baseline HLPS Detail

HLPS[31] is a hierarchical reinforcement learning method in which the high-level policy selects latent subgoals and the low-level policy produces actions to achieve them. The algorithm is based on Soft Actor-Critic[73], and we follow the original implementation with necessary modifications. Specifically, we adapt HLPS to support parallelized environments using Isaac Gym.

The state space for both the high-level and low-level policies is identical to that of the PPO baseline. However, the low-level policy additionally receives the latent subgoal sampled from the high-level policy as part of its input. The action space of the low-level policy is identical to that of the PPO baseline, where it outputs joint torque commands along with joint gain and damping values.

Both the high-level and low-level policies are trained with dense rewards. The high-level policy uses the same dense task reward as the PPO baseline, based on the distance between the current object pose and the target object pose. The low-level policy is trained with dense subgoal rewards, defined by the distance between the current state and the latent subgoal output by the high-level policy.

A key feature of HLPS is its latent subgoal representation. Instead of selecting subgoals directly in the object pose space, the high-level policy selects subgoals in a learned latent space structured by a probabilistic model using a Gaussian Process (GP). These latent subgoals are combined with the current state representation and provided to the low-level policy, which is trained to reach them.

Hyperparameters related to training HLPS are summarized in Table 23, and remain consistent across all tasks.

Hyperparameter	Value
Network Architecture	
Actor network hidden sizes	400, 400, 400, 400
Critic network hidden sizes	400, 400, 400, 400, 400, 400
Encoding network hidden size	100
Hidden layer activation (all networks)	ReLU
Actor output activation	Tanh
Optimizer	Adam
Training Parameters	
Latent GP learning rate	1×10^{-5}
Latent GP update frequency	10
Batch GP time window size	3
Latent subgoal dimension	7
Learning rate (actor/critic, both levels)	0.0002
High-level action interval k	50
Target network smoothing coefficient	0.005
Discount factor γ	0.99
Encoding layer learning rate	0.0001

Table 23: Hyperparameters for the baseline HLPS

C.2 Detailed Results of Simulation Experiments

The table 24 compares the number of state-action pairs (in billions) used for training across methods and domains. Notably, SPIN requires orders of magnitude fewer data samples than traditional RL-based methods like PPO, HLPS, and MAPLE, demonstrating its data efficiency. Skill-RRT is excluded from training as it is a non-learning baseline.

Table 25 provides a detailed version of the table 1 presented in Section 4, reporting the average success rate and computation time across three seeds on 100 test problems per domain. SPIN achieves the highest success rate across all domains, significantly outperforming other baselines, while maintaining reasonable computation time comparable to SPIN-w/o-filtering. In contrast, purely RL-based methods (PPO, HLPS) show zero success, highlighting the advantage of combining planning and learning.

Method	Card Flip	Bookshelf	Kitchen
PPO	2.5	2.5	2.5
HLPS	0.15	0.23	0.28
MAPLE	0.27	0.33	0.27
SPIN-w/o-filtering	0.0030	0.0028	0.0031
SPIN	0.0029	0.0028	0.0032

Table 24: Number of state-action pairs used for training across different methods and domains (in billions). Note that Skill-RRT does not require training, as it simply replays the first found skill plan in the simulator.

	Components			Problem Domain					
				Card Flip		Bookshelf		Kitchen	
	Method	Action Type	Use $\mathcal{K}$	Success rate (%)	Computation time (s)	Success rate (%)	Computation time (s)	Success rate (%)	Computation time (s)
PPO [32]	Flat RL	Low-level action	✗	0.0 ± 0.0	N/A	0.0 ± 0.0	N/A	0.0 ± 0.0	N/A
HLPS [31]	Hierarchical RL	Low-level action	✗	0.0 ± 0.0	N/A	0.0 ± 0.0	N/A	0.0 ± 0.0	N/A
MAPLE [24]	Hierarchical RL	Skill & Parameter	✓	0.0 ± 0.0	N/A	78.0 ± 2.4	5.3 ± 2.7	0.0 ± 0.0	N/A
Skill-RRT	Planning	Skill & Parameter	✓	39.0 ± 0.0	85.3 ± 48.7	66.0 ± 0.0	79.2 ± 67.1	64.0 ± 0.0	121 ± 39.5
SPIN-w/o-filtering	Planner distilled via IL	Low-level action	✓	82.0 ± 0.0	2.68 ± 0.61	83.0 ± 0.5	2.93 ± 2.05	87.0 ± 0.8	3.02 ± 0.65
SPIN	Planner distilled via IL	Low-level action	✓	95.0 ± 0.5	2.68 ± 0.61	93.0 ± 1.4	2.93 ± 2.05	98.0 ± 0.5	3.02 ± 0.65

Table 25: Comparison of baselines based on their components (method, action type, and whether use $\mathcal{K}$ or not) and performance metrics (success rate and computation time) with their average and standard deviation across three different seeds for each problem domain. The success rate is measured on a set of 100 test problems. Computation time refers to the average elapsed time required to solve the given 100 problems when the method successfully completes them.

C.3 Real-World Experiments Detail

In this section, we present a detailed setup for real-world experiments, covering the environment configuration and perception systems for three domains: card flip, bookshelf, and kitchen.

Environment Configuration We utilize the Franka Research 3 robot and its gripper across three domains. To enhance the gripper's effectiveness in non-prehensile manipulation, rubber is added to the gripper fingers to increase surface friction. The environment setups are illustrated in Figure 4.

- **Card Flip** For the real-world card flip domain, we construct a 30x30x40 cm table identical to the simulation table. The 5x7x0.5 cm card (cuboid shape) is textured with colors to break symmetry in its shape.
- **Bookshelf** In the real-world bookshelf domain, we build a bookshelf similar to the one used in simulation and utilize a 14x20x3 cm book (cuboid shape). To facilitate book toppling, sandpaper is affixed to the top surface of the book to enhance shear contact force.
- **Kitchen** The kitchen domain utilizes the IKEA DUKTIG play kitchen and a cup with a body that cannot be grasped. To enhance grasping stability, the cup handle is filled with foamboard.

Perception System The perception modules are used to estimate object poses in the real world. To address potential occlusions caused by the robot, multiple RealSense D435 cameras are installed, as shown in Figure 4. For object pose estimation, we use FoundationPose [74], an off-the-shelf RGB-D-based object pose estimator. During each episode, we select the camera with the best visibility, determined by the largest object segmentation mask. The object segmentation mask is initially generated using SAM [75] and subsequently updated over time using Cutie [76].

Figure 4: The real-world setup for each domain is illustrated, with blue polygons representing the target objects and red circles indicating the camera locations.

D Ablation Studies

In this section, we provide a detailed description of the ablation studies conducted to validate the design choices of our framework. Specifically, we investigate three aspects: (1) the necessity of learning the *connector* policy via reinforcement learning, by comparing it with a motion planner; (2) the effectiveness and importance of our *data filtering* method during imitation learning; and (3) the impact of different *model architectures* on imitation learning performance.

D.1 Connector Effectiveness Ablations

We need connector policies that bridge the state gaps between the termination of one skill and the initiation of the next. We use RL to train the connector policies, while motion planners (MP) could be used. There are two main reasons: (1) RL policies achieve higher success rates in bridging these gaps, especially during contact and de-contact between the robot and the object, and (2) neural network-based policies offer fast inference times, which help accelerate our planner, `Skill-RRT`. To demonstrate the necessity and robustness of our connectors, we compare them with MP, specifically Bi-RRT from OMPL [77].

The success rate of the MP is lower than that of the connectors in almost all cases, as shown in Table 26. In the Kitchen domain, Bi-RRT fails to find a valid joint trajectory from the non-prehensile's terminal state to the prehensile's initial state because the narrow sink space causes frequent collisions between the robot and the environment or the object. Additionally, our connectors offer significantly shorter inference times compared to the motion planner, which requires replanning for every problem instance.

Method	Card		Bookshelf		Kitchen	
	NP $\rightarrow$ P	P $\rightarrow$ NP	NP $\rightarrow$ P	P $\rightarrow$ NP	NP $\rightarrow$ P	P $\rightarrow$ NP
Connectors	91.0	88.0	99.0	94.0	90.0	78.0
Bi-RRT	80.0	86.0	55.0	61.0	0.0	81.0

Table 26: Comparison of the success rates between connectors and Bi-RRT on 100 connector tasks in simulation. Success indicates success rate among 100 problems with unit (%).

As illustrated in Fig. 5, failures occur in the following scenarios: (a) the robot cannot exactly follow the planned path, leading to the card being dropped during de-contact, (b) the object is not in a perfectly stable pose while the MP assumes a static scene, causing continued contact with the gripper even after the release, or (c) the robot is surrounded by obstacles at the initial state, and it becomes computationally expensive to sample a collision-free path. These cases motivate us to train connectors using RL. We strongly encourage referring to the supplementary video (MP), which illustrates these failure cases in action.

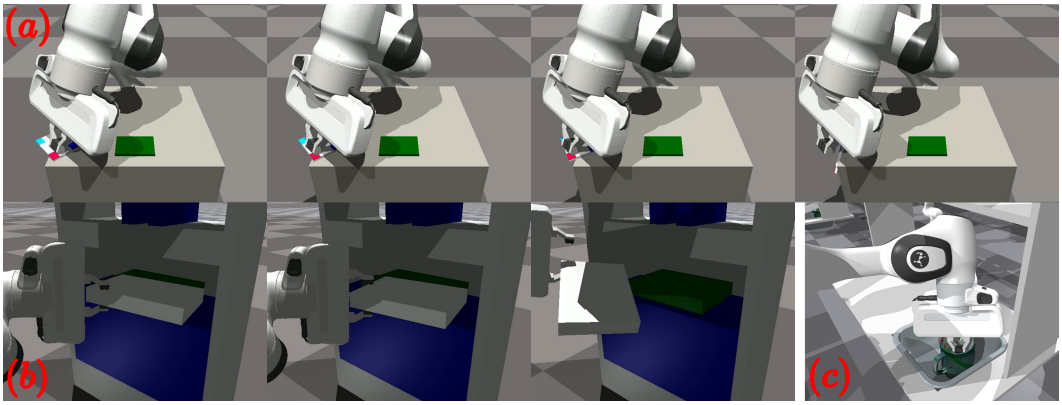

Figure 5: Illustration of motion planner failure cases.

D.2 Filtering Method

In this section, we study the impact of the replay success rate threshold parameter m and the effectiveness of our data filtering mechanism. As emphasized by Mandlekar et al. [67], the quality of the training dataset plays a critical role in the performance of imitation learning policies. We show that our filtering method significantly enhances data quality, leading to improved policy performance.

For the baseline (*Without Replay*), we collect 15,000 skill plans without applying any filtering, using one trajectory per skill plan to train the diffusion policy. In contrast, *Replay with* $m = 0.1$ and *Replay with* $m = 0.9$ filter out skill plans that do not meet their respective replay success rate thresholds. The replay success rate of a skill plan is defined as

$$\frac{\text{Number of successful replays of } \tau_{\text{skill}} : N_{success}}{\text{Total number of replays of } \tau_{\text{skill}} : N}$$

. A skill plan is included in *Replay with* $m = 0.1$ if $N_{\text{success}} > 40$, and in *Replay with* $m = 0.9$ if $N_{\text{success}} > 360$.

For both replay settings, we collect 500 skill plans. From each skill plan, we randomly sample 30 successful trajectories, resulting in a total of 15,000 trajectories per method to ensure a fair comparison during training. After collecting the trajectories, we train the distillation policy (Diffusion Policy [5]) using the datasets filtered by each method via IL. The training and evaluation procedures are identical to those described in Appendix D.3.

Table 27 compares three different filtering methods: *Without Replay*, *Replay with* $m = 0.1$, and *Replay with* $m = 0.9$ (ours). The table summarizes key metrics, including the number of skill plans collected, the number of trajectories generated per skill plan, and the success rates achieved across different domains. As shown, our method (*Replay with* $m = 0.9$) consistently achieves the highest success rates.

We further analyze the collected skill plans from each method in two ways. First, we examine the distribution of replay success rates, as shown in Figure 6. Second, we investigate the desired object poses in the skill plans across the Card Flip, Bookshelf, and Kitchen domains, as presented in Figures 7, 8, and 9. Detailed explanations for each figure follow in the subsequent paragraphs.

Figure 6 compares the replay success rates of skill plans collected by each filtering method. The y-axis shows the frequency of skill plans, and the x-axis represents their replay success rate. All success rates are measured by replaying each skill plan $N = 400$ times in the GPU-based simulator, IsaacGym. Filtering with $m = 0.9$ prioritizes high-quality skill plans by selecting those with a higher probability of successful replay. As a result, skill plans collected with $m = 0.9$ exhibit a significantly higher average replay success rate compared to other methods.

The left and center charts of Figure 7 show the distribution of the absolute y-position of the desired card pose for the slide skill, which is executed before the prehensile skill in the skill plan, extracted

	Characteristic		Data Collection		Domain name		
					Card Flip	Bookshelf	Kitchen
	Replay	$m = 0.9$	Skill plans	Traj per a plan	Success rate (%)	Success rate (%)	Success rate (%)
Without Replay	✗	✗	15000	1	79	95	86
Replay with $(m = 0.1)$	✓	✗	500	30	82	83	87
Ours (*Replay with* $(m = 0.9)$)	✓	✓	500	30	**94**	**96**	**98**

Table 27: Comparison of three different filtering methods: *Without Replay*, *Replay with* $m = 0.1$, and Ours (*Replay with* $m = 0.9$). The table shows several metrics, including the characteristics, the number of skill plans collected, the number of trajectories used for training from each skill plan, and the success rates for each domain across the three filtering methods. "Skill plans" refers to the number of skill plans for each filtering method. "Traj per a plan" refers to the number of successful trajectories used for training from each skill plan. "Success rate" refers to the task success rate of the distillation policy, which is trained with data from each filtering method.

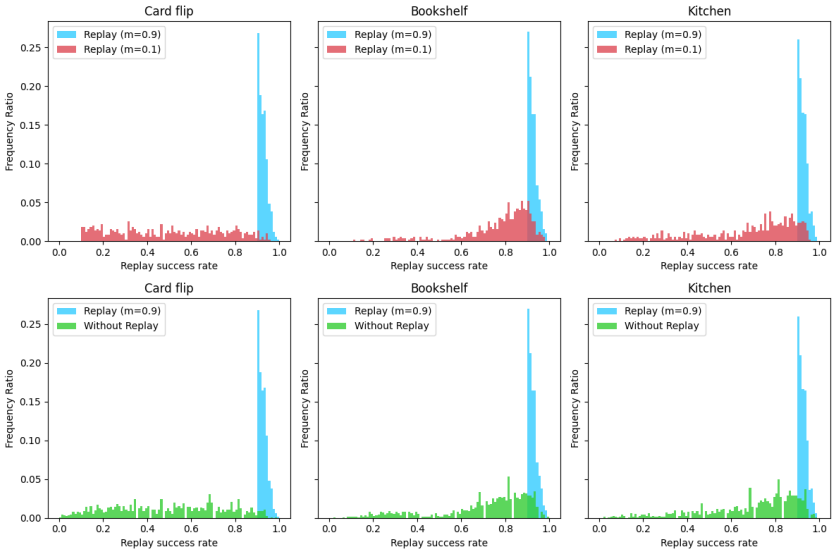

Figure 6: Compare collected skill plan's replay success rate for each filtering method

from skill plans collected by each filtering method. The y-axis shows the frequency, and the x-axis represents the absolute y-position. For successful grasping, the card should be placed near the table edge ($y = 0.15$ or -0.15), but placing it too close increases the risk of falling. Thus, the ideal y-position is slightly smaller than 0.15, balancing graspability and stability. The right chart of Figure 7 confirms that skill plans with higher success rates position the card further inward, reducing failure risk while maintaining graspability.

The left and center charts of Figure 8 show the distribution of the θ_z (yaw) angle of the desired book pose for the topple skill. The y-axis indicates frequency, and the x-axis represents the θ_z angle. A larger toppling angle increases the risk of the book falling, leading to task failure, as shown in the right chart. Skill plans with higher success rates constrain the topple angle to smaller values, ensuring stability, whereas plans with lower success rates allow larger angles, increasing instability.

The left and center charts of Figure 9 show the distribution of the θ_z (yaw) angle of the desired cup pose for the sink skill. The y-axis shows frequency, and the x-axis represents the θ_z angle.

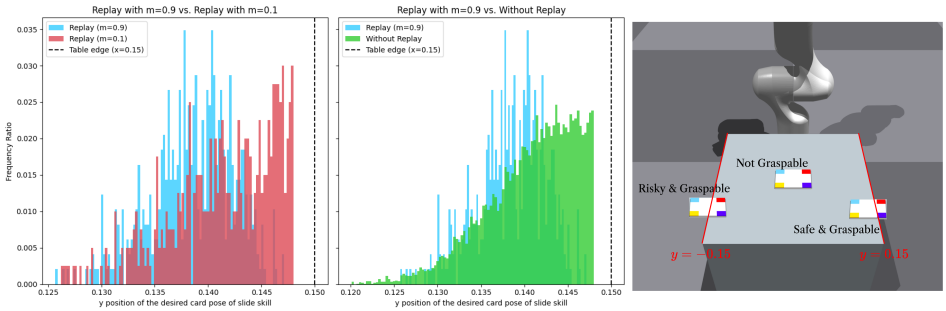

Figure 7: The charts on the left and center compare the absolute value of the y-position of the desired card pose for the slide skill (before the P skill) across each filtering method. The chart on the right shows the graspability of the card pose based on the y-position in the simulation.

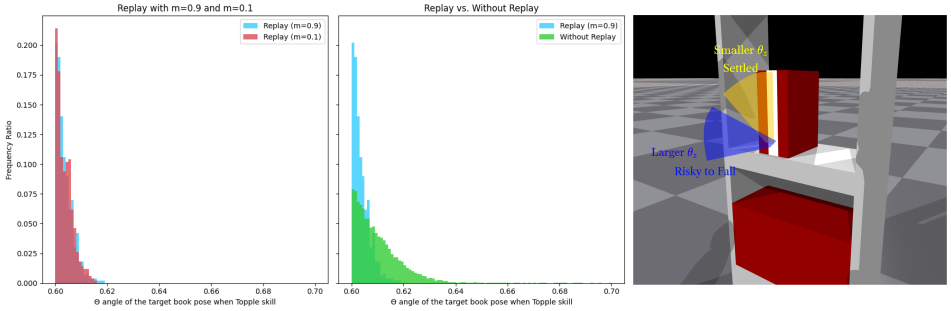

Figure 8: The charts on the left and center compare the θ_z (yaw) angle of the desired book pose for the Topple skill across each filtering method. The chart on the right shows that higher values of θ_z are riskier for the book to fall down in the simulation.

For efficient execution, the cup's handle should ideally point toward the robot within $[\pi/2, 3\pi/4]$. If the handle is not properly oriented, grasping becomes physically infeasible: when the handle faces away, the end-effector cannot reach the grasp point due to collisions with the underside of the cupboard. Conversely, even if the handle faces the robot directly, it may still be difficult to find a collision-free grasp pose because of the robot hardware constraints, resulting in inverse kinematics (IK) failure. The right chart shows that skill plans with higher success rates fall within the optimal range, whereas lower-quality plans cover a broader, suboptimal range.

While filtering with a high replay success threshold ($m = 0.9$) leads to robust distillation policy performance, it also increases collection time, as shown in Table 28. Many skill plans are discarded for not meeting the $m = 0.9$ criterion, resulting in longer collection time per plan. To mitigate this, we generate 30 successful trajectories per skill plan by replaying, leveraging the simulator's stochasticity and observation noise from domain randomization. This approach is more time-efficient than collecting a large number of skill plans, as done in the Without Replay setting.

D.3 Imitation Leaning Policy Architectures

In this section, we study how the policy architecture choices in IL affect the performance of the distillation policy in simulation. Specifically, we compare the diffusion policy [5] with four alternative architectures: (1) ResNet [78], a simple IL model with a large parameter size; (2) LSTM+GMM, which has been shown to be effective for multimodal data in RoboMimic [79]; (3) cVAE [80], another conditional generative model; and (4) Transformer [81], a widely used architecture for multimodal data such as language. Each architecture is trained on the same dataset, with approximately 70 million parameters, for 100 epochs, across three different seeds.

For training, we adapt the IL codebase from RoboMimic [79]. The MLP backbone is replaced with ResNet for a fair comparison with other large models. We do not modify the original code of

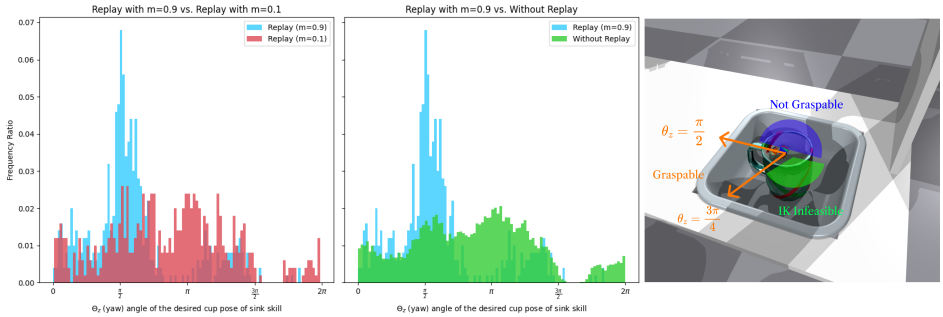

Figure 9: The charts on the left and center compare the θ_z (yaw) angle values of the desired cup pose for the Sink skill across each filtering method. The chart on the right shows the graspability based on the θ_z (yaw) angle of the Sink skill's desired cup pose in the simulation.

	Data Collection		Card Flip			Bookshelf			Kitchen		
	Skill plans	Traj per a plan	Time per plan (s)	Total time (s)	Discarded plans	Time per plan (s)	Total time (s)	Discarded plans	Time per plan (s)	Total time (s)	Discarded plans
Without Replay	15000	1	85.3	$1.28 * 10^6$	0	79.2	$1.19 * 10^6$	0	121.0	$1.82 * 10^6$	0
Replay with $(m = 0.1)$	500	30	89.5	$4.47 * 10^4$	23	79.5	$3.98 * 10^4$	2	122.6	$6.13 * 10^4$	6
Ours (*Replay with* $(m = 0.9)$)	500	30	2372.1	$1.18 * 10^6$	13403	449.4	$2.2 * 10^5$	2437	1048.6	$5.2 * 10^5$	3933

Table 28: Data collection time for each filtering method. "Skill plans" refers to the number of skill plans for each filtering method. "Traj per a plan" refers to the number of successful trajectories collected for each skill plan. "Time per plan" indicates the time taken to collect a skill plan using the planner `Skill-RRT`. "Total time" represents the total collection time required to gather the specified number of skill plans under each filtering method. "Discarded plans" denotes the number of skill plans discarded due to failing to meet the replay success rate criteria ($m = 0.1$ or $m = 0.9$).

LSTM+GMM, cVAE, or Transformer, except for adjusting hyperparameters such as model size. The hyperparameters, which are kept consistent across the three domains, are summarized in Table 29. Each architecture is trained for 100 epochs.

Common			
State Normalization	Z-Score	Action Normalization	Min-Max
Batch size	2048	Epochs	100
Optimizer	AdamW	Learning Rate	1e-4
Weight Decay	0.1	LR Scheduler	Cosine
ResNet		**LSTM+GMM**	
MLP Dimensions	[4096x3, 2048x7, 1024]	History Length	10
Activation Function	GELU	LSTM Dimensions	[1470x4]
Loss	L2 Norm	Last Layer Dimension	2048
		Activation Function	GELU
		Number of GMM Modes	10
		Loss	NLL
cVAE		**Transformer**	
Encoder Dimensions	[1024, 1024, 2048]	Embedding Dimension	1024
Decoder Dimensions	[1024, 1024, 2048]	Number of Layers	6
Prior Dimensions	[1024, 1024, 2048]	Number of Heads	8
Latent Dimensions	14	Dropout	0.1
Loss	VAE	Activation	GELU
KL Loss Weight	1.0	Loss	L2

Table 29: Hyperparameters of IL policy architectures

	Card Flip	Bookshelf	Kitchen
ResNet [78]	54 ± 5.7	85 ± 2.6	98 ± 0.5
LSTM+GMM [79]	55 ± 5.7	95 ± 1.7	$\mathbf{99 \pm 0.5}$
cVAE [80]	22 ± 2.1	31 ± 3.3	41 ± 4.5
Tranformer [81]	63 ± 1.4	$\mathbf{96 \pm 1.4}$	98 ± 0.8
Diffusion Policy [5]	$\mathbf{95 \pm 0.5}$	93 ± 1.4	98 ± 0.5

Table 30: Success rates with unit (%) averaged over 3 seeds in simulation for each IL policy architectures.

The success rate is measured on 100 fixed test problems, which specify the initial object configuration $s_0.q_{\text{obj}}$, the initial robot configuration $s_0.q_{\text{r}}$, and the goal object configuration q^g_{obj}. We record the success rate at every epoch, and the performance of each architecture is evaluated based on the maximum success rate achieved during training across three seeds. The results are summarized in Table 30.

In the card flip domain, the diffusion policy significantly outperforms the other architectures. We hypothesize that this advantage stems from the high diversity of intermediate target poses in this domain, where the object poses can vary between, for example, the left and right edges of the table. In contrast, in the bookshelf and kitchen domains, ResNet, LSTM+GMM, and Transformer achieve success rates comparable to the diffusion policy, possibly due to the lower diversity of intermediate poses. cVAE exhibits consistently poor performance across all domains, suggesting that the smoothing effect of VAE impairs IL performance. These results indicate that the overall success of our framework is not solely attributed to the choice of the diffusion policy. Although the diffusion policy achieves slightly higher performance, other architectures such as ResNet, LSTM+GMM, and Transformer also demonstrate strong success rates. This suggests that our framework's effectiveness is not dependent on a specific policy architecture, but rather that the diffusion policy was selected as the optimal choice among several viable options.