A Long Horizon Planning Framework for Manipulating Rigid Pointcloud Objects

Anthony Simeonov[1], **Yilun Du**[1], **Beomjoon Kim**[1,2], **Francois R. Hogan**[1,3],
Joshua Tenenbaum[1], **Pulkit Agrawal**[1], **Alberto Rodriguez**[1]
[1]Massachusetts Institute of Technology — {asimeono,yilundu,jbt,pulkitag,albertor}@mit.edu
[2]KAIST Graduate School of AI — beomjoon.kim@kaist.ac.kr
[3]Samsung AI Center Montreal — f.hogan@samsung.com

Abstract: We present a framework for solving long-horizon planning problems involving manipulation of rigid objects that operates directly from a point-cloud observation. Our method plans in the space of object subgoals and frees the planner from reasoning about robot-object interaction dynamics. We show that for rigid-bodies, this abstraction can be realized using low-level manipulation skills that maintain sticking-contact with the object and represent subgoals as 3D transformations. To enable generalization to unseen objects and improve planning performance, we propose a novel way of representing subgoals for rigid-body manipulation and a graph-attention based neural network architecture for processing point-cloud inputs. We experimentally validate these choices using simulated and real-world experiments on the YuMi robot. Results demonstrate that our method can successfully manipulate new objects into target configurations requiring long-term planning. Overall, our framework realizes the best of the worlds of task-and-motion planning (TAMP) and learning-based approaches. Project website: https://anthonysimeonov.github.io/rpo-planning-framework/.

Keywords: Manipulation, Learning, Planning

1 Introduction

Consider the bi-manual robot in Figure 1 tasked with moving a block (red) into a target pose on the far side of the table (green). The robot has a library of manipulation skills: *pull*, *push* and *grasp-reorient* (flips the block so that it rests on a different face). To move the block the robot must first pull the object close to the center of the table, where it can be grasped with both palms to re-orient it. After reorientation, due to the robot's limited reach, the block cannot be directly placed at the target position. Instead, it must first be placed at a point within reach of both the manipulators and then pulled by the left hand to the target. Our goal is to develop a robotic system that addresses the perceptual and planning challenges of such long-term tasks requiring coordinated and sequential execution of multiple skills on a variety of objects of different shapes and sizes. The approach we propose is guided by two key observations:

- The challenge of **planning for long-term tasks** arises from a large search space. Instead of reasoning in the space of low-level robot actions, the search space can be substantially reduced by planning in the more *abstract* space of object configurations (or, *subgoals*) and manipulation skill parameters (or, *contact locations*) – an idea well studied in Task-and-Motion planning (TAMP) [1, 2, 3, 4]. However, each skill imposes a skill-specific relationship between *reachable* object subgoals, *compatible* robot-object contacts, and *feasible* robot motion. For instance, all object subgoals cannot be realized by the robot due to kinematic/dynamic or other constraints, and only certain contact locations on the object enable achieving any particular subgoal. As a result, decoupling *what* the subgoal is and *how* the robot reaches it typically leads to computational inefficiencies in searching through many infeasible plans. We mitigate some of these inefficiencies by learning models that capture this coupled relationship between subgoals and skill parameters.
- The complexity of **perceiving unknown objects of a variety of shapes and sizes** arises due to the unavailability of 3D object models and difficulties in obtaining an object's full geometry/state

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

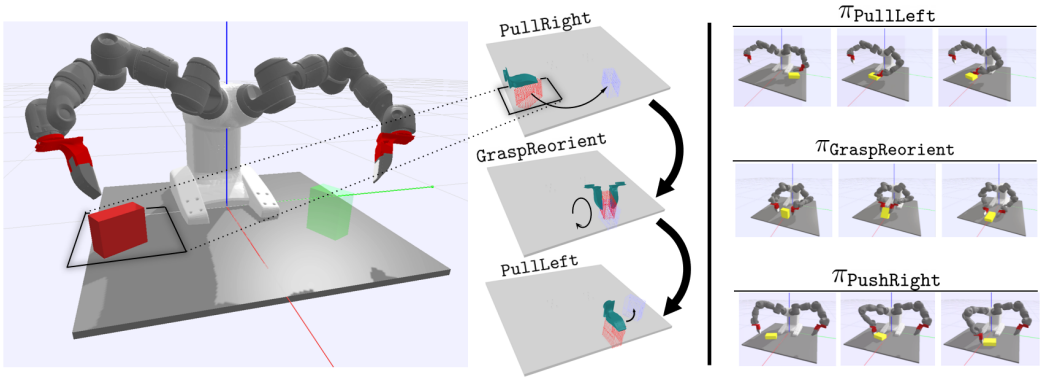


Figure 1: **Left**: Our framework uses learned samplers and a family of primitive manipulation skills to imagine and execute multi-step plans that manipulate objects between stable poses, using only a segmented object point-cloud. First (top right), our learned model for *pull* samples a palm pose (dark green) and a subgoal (blue), when provided with an input point-cloud (red). The *pull* subgoal becomes the input at step two for *grasp-reorient*. Sampling repeats iteratively toward the overall goal (green). **Right**: Examples executions of the manipulation skills from [9] we use in our setup.

from noisy sensor observations. These challenges suggest that we bypass attempting to infer the full object state and instead build perception models that process observations in direct support of predicting skill parameters that are likely to succeed, a technique that is natural to affordance prediction frameworks [5, 6, 7, 8]. In particular, for *every skill*, we learn a separate *skill parameter sampler* that predicts a distribution over *subgoals* and *contacts* directly from an object point-cloud.

Conventional TAMP methods usually tackle the challenges mentioned above by introducing hand-designed heuristics for sampling parameters. However, these heuristics are typically based on *a priori* information about the object's shape, limiting their utility when attempting to manipulate novel objects. On the other hand, past works that make predictions directly from sensory observations overcome the problem of generalization to new objects [10, 11, 12], but face difficulty in long-horizon tasks due to operating in the space of low-level actions.

At the intersection of these domains lies our central problem, namely, to enable *efficient* planning over high-level sequences of *object subgoals* and *skill parameters* from *sensory observations*. The complexity now is in generating sequences of object subgoals that that are compatible with the family of manipulation skills, and obtaining feasible robot actions that are suitable to reach these subgoals. We alleviate some of this complexity by assuming that (a) objects are rigid; and (b) sticking contacts occur between the robot and the object. These assumptions enable reasoning about actions purely at the level of contact configurations between the manipulator and the object. This is because: (i) subgoals can be represented as sequences of $SE(3)$ transformations; and (ii) sticking contact enforces a fixed rigid transformation between the robot and the object during interaction. Recent work by Hogan et al. [9] proposes a set of manipulation skills that exploit rigid sticking contacts, and operate from an initial contact configuration with the object to produce a desired rigid transformation. Their approach to planning and perception works in state space and requires the object's 3D model.

Technical contributions In this work we leverage deep learning techniques to generalize these skills to work directly with segmented point-clouds. We learn efficient samplers trained to map an object point-cloud to compatible distributions of both reachable subgoals (in the form of rigid object transformations) and affordances of contact locations (in the form of Cartesian poses of the end effector). Integrating these samplers into our framework allows us to combine the strengths of TAMP search and learned affordances, to construct long-term plans that also generalize to novel objects.

Our contributions are two-fold: (i) a planning and perception framework for sequential manipulation of rigid bodies in point-cloud space with manipulation skills that exploit rigid sticking interaction to enable rigid object transformations; (ii) specific architectural choices that significantly improve planning performance and efficiency. These choices are: (a) an object/environment geometry-grounded representation of reorientation subgoals for neural networks to predict; (b) a novel graph-attention based model to encode point-clouds into a latent feature space; (c) and joint modeling of what subgoals to achieve and how to achieve them, instead of modeling them independently.

Results We validate our approach using simulated and real-world experiments on a bi-manual YuMi robot that manipulates objects of previously unseen geometries into target configurations by sequencing multiple skills. We show our method outperforms carefully designed baselines that utilize privileged knowledge about object geometry and task in terms of planning efficiency, and we present detailed ablation studies to quantify the performance benefits of our specific system design choices. Overall, the proposed framework aims to combine the strengths of TAMP and learning-based manipulation methods. On one hand, we generalize traditional TAMP approaches to work with perceptual representations and previously unseen objects. On the other hand we are able to perform long-term planning which remains challenging for end-to-end learning based systems.

2 Problem Setup and Multi-step Planning

Assumptions Our work makes three key assumptions: 1) Objects are rigid and are represented as a segment of a point-cloud, which we refer to as *Rigid Pointcloud Objects*; 2) Subgoals are represented as sequences of $SE(3)$ transformations of the object point-cloud; and 3) A fixed rigid relationship between the robot and object is maintained during interaction, a practice that is common to manipulation approaches that exploit sticking contacts [13, 9]. We also assume quasi-static interactions, segmentation of the entire point-cloud to obtain the object-point cloud, and a high-level plan skeleton [14] that specifies the sequence of skill *types* required to solve the task. Note that the problem of realizing a plan skeleton is addressed by existing orthogonal works [15, 16, 17].

Task Setup The robot is tasked to transform a rigid object by an $SE(3)$ transformation denoted by T^o_{des}, and is provided with a segmented point-cloud observation of the object, $X \in \mathbb{R}^{N\times 3}$. We assume that reaching the target configuration requires a T step plan. The planner must determine a sequence of object transformations, $T^o_{1:T} \in SE(3)$ and a sequence of poses of the left and right manipulator when they make contact with the object denoted as $T^{p_c}_{1:T} = \{T^{p_c}_{R_{1:T},L_{1:T}}\} \in SE(3)$, where $T^{p_c}_{R_i,L_i}$ denotes the pose of the left and right manipulator respectively at the i^{th} step of the plan. If the plan is successful, then $\prod_{i=1}^{T} T^o_i = T^o_{des}$. For rigid-body manipulation, planning in the space of (T^{p_c}, T^o) is sufficient if the robot maintains a sticking contact with the object when it is being manipulated. Sticking contact ensures that desired object transformation T^o_i is also the transformation that the manipulators have to undergo after having made contact with the object. We denote T^{p_c} as a contact pose and T^o as a subgoal.

Manipulation Skills In general it is challenging to find a motion-plan (i.e., a sequence of robot joint configurations) that manipulates the object while maintaining sticking-contact. We borrow from the work of [9] a set of manipulation skills $\{\pi^1, ..., \pi^K\}$ that transform the object while maintaining sticking contact. We use $K = 6$, which corresponds to skills of pulling (R/L), pushing (R/L), pick-and-place, and re-orientation [9] (Figure 1 Right). The choice of these skills constrains the planner to only consider T^o_i that are achievable by one of the K skills. Given the (T^{p_c}, T^o) determined by the planner, each skill can be thought of as low-level planner,

$$\mathbf{q}_{R,L} = \pi(T^{p_c}, T^o) \tag{1}$$

that outputs a sequence of joint configurations of the right $(\mathbf{q}_R)$ and left $(\mathbf{q}_L)$ manipulators required to transform the object by T^o. The combined joint sequence denoted by $\mathbf{q}_{R,L} = \{q_0, ..., q_F\}$ is of length (F), which is variable and depends on the skill-type and the inputs to π. The skill is feasible if $\mathbf{q}_{R,L}$ is collision-free, respects joint limits, and avoids singularities.

Planning We assume that the robot is given a plan skeleton $PS = \{S_t\}^T_{t=1}$, where $S_t \in \pi^{1:K}$ denotes the skill-type. Given the plan skeleton, the planner needs to determine for every skill the: (a) object subgoal transformation (T^o_t) and (b) contact poses $(T^{p_c}_t)$. We use a sampling-based planner to search for sequences of object transformations and contact points $\{T^o_{1:T}, T^{p_c}_{1:T}\}$. In our framework, each skill (S_t) in the plan skeleton (PS) can be thought of as a node. The goal of the planning is to connect these nodes. Application of skill S_t at t^{th} node, transforms the input point-cloud X_t into the next point-cloud $X_{t+1} = T^o_t X_t$. To efficiently sample plausible plans, it is necessary to only sample T^o_t that can be achieved by the S_t given the current object configuration X_t. For this, we learn a sampler for the contacts and subgoal transformations, $(T^{p_c}_t, T^o_t)$, using a neural network (NN), denoted $p_{\pi_t}(\cdot|X_t;\theta)$, where θ are weights of the NN. We discard samples $(T^{p_c}_t, T^o_t)$ that are infeasible. For the final step in PS, T^o_T can be solved for based on the required transformation-to-go, as $T^o_T = T^o_{des}\big(\prod_{t=0}^{T-1} T^o_t\big)^{-1}$. Sampling along PS continues until we find a feasible sequence $(T^{p_c}_{1:T}, T^o_{1:T})$ or the planner times out. See appendix for the full algorithm.

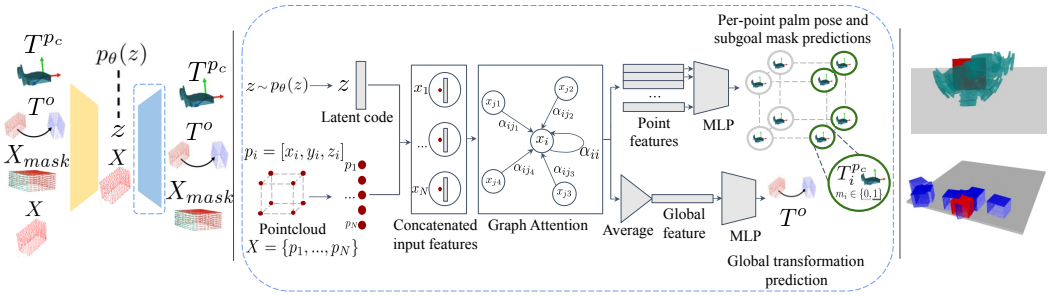


Figure 2: **Left**: CVAE encoder-decoder structure. The encoder maps the data to a conditional latent space, which is constrained to match a prior. The decoder takes as inputs a random vector z from the latent space and point-cloud X, and outputs the contact poses, object subgoal, and a mask, denoted X_{mask}, that indicates which part of the object must be in contact with the resting surface. **Middle**: Decoder architecture. Point-cloud point features are concatenated with a latent code, and encoded with multiple layers of self-attention into per-point output features and an average-pooled global feature. Output point features are used to predict separate contact poses and a sub-goal mask. The global feature is used to predict the subgoal **Right:** Pulling T^{p_c} (top) and T^o (bottom) samples

3 Learning the Skill Sampler

Our method represents skill samplers $\{p_{\pi^i}(\cdot|X;\theta)\}_{i=1}^K$ using a conditional variational autoencoder (CVAE; [18]) that predicts a distribution over skill parameter (T^{p_c}, T^o) values conditioned on the point cloud observation. For training, we collected a dataset D_{π^i} consisting of successful single-step skill executions, $D_{\pi^i} = \{(X^{(i)}, T^{o\,(i)}, T^{p_c\,(i)})\}_{i=1}^n$. The details of the CVAE are provided in the appendix. The specific modelling choices that were critical for accurate manipulation and generalization to unseen objects are described in the following subsections.

Learning Pointcloud Features for Predicting Contact Poses Regressing the end-effector pose at which the robot contacts the object point cloud T^{p_c} and the subgoal T^o requires learning a good point-cloud feature representation. Prior work made use of methods such as PointNet++[19] to learn features for successfully predicting end-effector pose [20, 21, 22, 23]. However, due to their construction, PointNet++ style architectures only coarsely model the object geometry which resulted in poor performance in our setup where accurate alignment of left and right grippers is necessary to manipulate the object. Instead, we find GAT to capture more rich geometric shape and state information, due to how the architecture explicitly models the relations between points in the input. Our architecture for the decoder, which outputs contact end-effector poses and the object subgoal, is shown in Figure 2 (Middle). Each point is treated as a node in a fully-connected graph, and at the input stage is represented by a concatenated feature vector: $(p_j \oplus z); j \in [1, N]$, where z is a sample from the learned latent distribution and p_j is the 3D coordinate of the j^{th} point. After processing through multiple layers of self-attention, we use two separate branches with fully-connected layers to predict T^{p_c} and T^o respectively. We found that while it was sufficient to average pool the features of all nodes to predict T^o, predicting T^{p_c} for each node in the graph separately was beneficial during training. This introduces an information bottleneck that helps learn more useful point-cloud feature extractors. At test time, the average of T^{p_c} predictions for all the nodes is used for execution.

Obtaining Accurate Subgoal Prediction For predicting T^o, it is worthwhile to note that while some skills such as *pull*, are designed for moving objects in $SE(2)$, other skills such as *grasp-reorient* reorients the object in $SE(3)$. While we could train a model to directly predict $SE(3)$ transformation representing T^o, we find this frequently leads to inaccurate predictions for skills designed to reorient the object. Using an inaccurate T^o to forward propagate the point-cloud

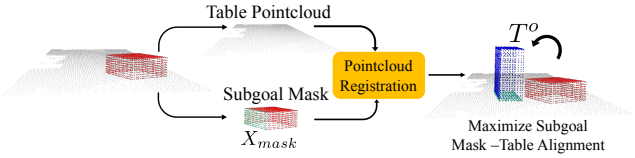


Figure 3: Illustration of reorientation subgoal prediction. A segmentation mask X_{mask} (green) of points on the object pointcloud that should contact the table after executing the skill is predicted by the neural network. A object transformation T^o is obtained through point-cloud registration of X_{mask} onto the table surface.

leads to accumulating errors which in turn leads to poor planning performance. We observed that inaccurate predictions of T^o corresponded to physically unrealizeable configurations such as object floating in air, penetrating the table, etc. Figure 5c shows examples. We resolve this problem by leveraging the insight that predicting the reorientation subgoal can be recast as the problem of selecting *which points on the object should end in contact with the support surface*. To implement this, we use the planar translation component of T^o, but compute the orientation component with following procedure. For each point on the object point-cloud, our model predicts per-point probabilities that are thresholded to obtain binary labels $m_j \in \{0, 1\} \forall j \in [1, N]$, that encode a segmentation mask X_{mask}. Once we have this mask, we can use point-cloud registration between the table point-cloud X_{table} and X_{mask} to solve for the orientation (see Figure 3), and combine it with the translation component of T^o. Because the errors in predicting the mask are averaged across the points when performing the registration step, it is more robust to prediction errors. It therefore leads to a more accurate prediction of T^o.

Modeling the joint distribution of Subgoals and Contact Points We found that predicting the subgoals and contact poses (T^o, T^{p_c}) independently often leads to incompatibility between the parameters. For instance, a sampled T^{p_c} can result in a good contact, but the motion required to achieve the independently sampled subgoal T^o results in a collision with the table. We mitigate this issue by modelling the joint distribution over these parameters by training the CVAE encoder to learn a single shared latent distribution. The decoder uses this latent representation to predict T^o, T^{p_c} and X_{mask} using separate output heads.

Training Data The training data D is generated by manipulating cuboids of different sizes using one primitive skill at a time. For data collection, we sample cuboids so that they are in a stable configuration on the table. We then randomly sample subgoals $(T^{o\ (i)})$ and contact points on the object mesh. Assuming the knowledge of 3D models of the cuboids, we execute a single primitive skill [9]. If the skill successfully moves the object to the subgoal, we add the tuple of (object pointcloud, contact points, subgoal and X_{mask}) to the dataset. We used manipulation data from 50-200 different cuboids for each skill and our overall dataset contained 24K samples. We trained separate CVAEs with the same architecture for each skill. See the appendix for more information on data generation, training, and network architecture.

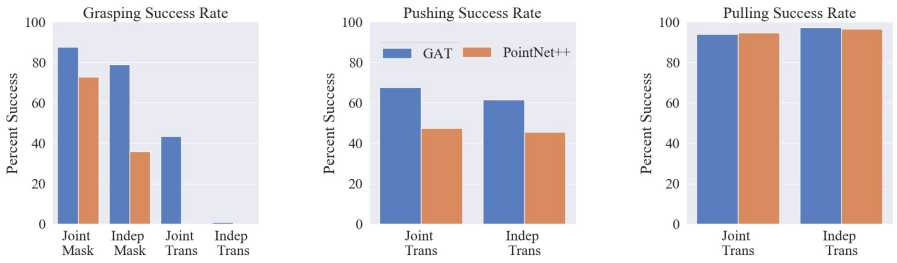
4 Experiments and Results

We perform experiments on a dual arm ABB YuMi robot. The robot has palm end-effectors, and is simulated in a table-top environment simulated in PyBullet [24] using the AIRobot library [25]. We place 4 RGB-D cameras with a shared focal point at the table corners, and we use ground truth segmentation masks to obtain segmented point-clouds from simulated depth images.

4.1 Performance Evaluation on Single-Step Manipulation Tasks

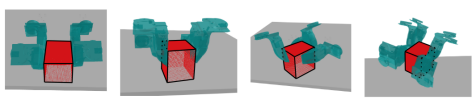
We first evaluate the efficacy of various design choices used to construct the sub-goal and contact point sampler on the task of single-step manipulation. For this we used 19 unseen cuboids of different sizes. To generate evaluation data, we initialized each of these cuboids in 6 uniformly and randomly sampled poses in a manner that the cuboid rested on the table. Given the point-cloud observation of the object, X, we predicted T^o, T^{p_c} from the learned sampler. If the sampler was accurate, then it means that execution of $\pi(T^o, T^{p_c})$ (see Equation 1) should transform the object by T^o. After execution we use the simulator state to evaluate the error between the desired (T^o) and actual object transformation. We define the *success rate* as the the percentage of trials where the algorithm finds feasible samples within 15 attempts *and* the executed object transformation is within a threshold of (3 cm/20°) of the desired subgoal.

We first compare the performance of PointNet++ [19] against the GAT architecture proposed in Section 3 to construct the CVAE encoder and decoder. Results in Figure 4 shows that GAT (blue) outperforms PointNet++ (orange) on grasping and pushing. The performance of both architectures is similar for pulling because it's a much easier manipulation to perform. To gain further insights into differences between the performance of GAT and PointNet++, we visualized the contact poses predicted by both the models in Figure 5a and 5b respectively. It can be seen that predictions from PointNet++ are more likely to result in robot's palms not aligning with the object (see Figure 5b).

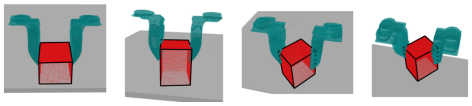


(a) (b) (c)

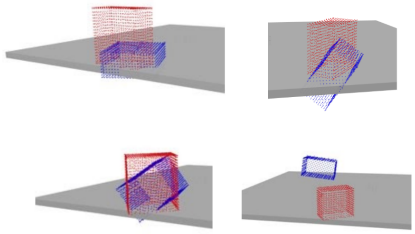
Figure 4: Success rates for single step grasping, pushing, and pulling. Joint Mask refers to using X_{mask} and predicting T^o and T^{p_c} together, Indep Mask refers to using X_{mask} but predicting T^o and T^{p_c} separately, and Joint Trans refers to not using X_{mask} and predicting T^o and T^{p_c} together. The overall results indicate a large value provided by using our point-cloud encoder, mask subgoal prediction, and joint prediction scheme. Please see the appendix for more detailed results breakdown.



(a) Predicted grasps from GAT (ours).



(b) Predicted grasps from PointNet++.



(c) Result of directly predicting T^o.

Figure 5: Visualization of predicted grasps from GAT and PointNet++ are presented in rows (a) and (b) respectively. (c) Using the skill sampler to directly predict T^o leads to inaccurate predictions, because small errors in T^o can correspond to physically infeasible configurations (blue point-clouds). Our proposed approach using a mask-based sub-goal representation overcomes this issue.

This results in less stable point contacts at edges instead of patch contacts on faces, which results in inaccurate or infeasible manipulation. Contact with edges also leads to object motion while the robot is trying to grasp. This breaks the sticking-contact assumption and consequently leads to inaccurate transformation of the object.

Next, for the *grasp-reorient* skill, we evaluate if predicting the mask-based subgoal (see Section 3) indeed achieves better performance than directly predicting T^o. Results in Figure 4a justify this choice. Qualitative examples of the type of errors made by directly predicting T^o are visualized in Figure 5c. It can be seen that these examples correspond to physically infeasible transformations, an error-mode which is fixed by using analytical computation of T^o using the mask-based representation. In contrast to *grasp-reorient*, *pull* and *push* skill subgoals only operate in $SE(2)$ and therefore do not suffer from this error mode. For these skills, direct prediction of T^o suffices for good performance.

Finally, we also contrast the performance of our method of jointly modelling subgoals and contacts against a baseline that models them independently. Results in Figure 4 shows that joint modelling leads to significant performance gains.

4.2 Performance Evaluation on Multi-step Manipulation Tasks

We now evaluate the performance of our planning algorithm (Section 2) when coupled with either our learned skill samplers or with a set of manually designed samplers. The experiment is designed to test the hypothesis that our learned samplers enable efficient multi-step planning with unseen objects for a variety of reconfiguration tasks. The baseline samplers use a combination of heuristics (plane segmentation, antipodal point sampling, and palm alignment with estimated point-cloud normals) based on the privileged knowledge the objects are cuboids (see appendix for details).

We consider multi-step planning problems on a set of 20 novel cuboidal objects, and evaluate the following metrics for a variety of plan skeletons: (i) planning success rate with a fixed planning budget of 5-minutes, (ii) average final pose error, and (iii) average planning time required over all

Table 1: Comparing the multistep planning success rate, average planning time, and error in achieving the target configuration between learned and hand-designed samplers. Values were obtained over 200 trials, with error bars denoting a 95% confidence interval. `P`: `PullRight`, `G`: `GraspReorient`

Type	Skeleton	Success Rate (%)	Pose Error (cm / deg)	Time (s)
Learned	P→G	96.9 ± 2.3	0.7 ± 0.5 / 3.0 ± 1.3	36.0 ± 6.9
	G→P	95.9 ± 2.8	3.8 ± 0.4 / 28.2 ± 4.9	30.5 ± 7.1
	P→G→P	86.9 ± 4.8	2.2 ± 0.4 / 18.9 ± 3.9	63.7 ± 10.6
Hand Designed	P→G	32.2 ± 6.5	0.9 ± 0.4 / 8.3 ± 5.0	110.0 ± 19.8
	G→P	70.4 ± 6.3	3.2 ± 0.5 / 18.1 ± 5.2	65.4 ± 12.9
	P→G→P	54.3 ± 7.0	2.4 ± 0.8 / 16.2 ± 6.5	69.7 ± 15.1

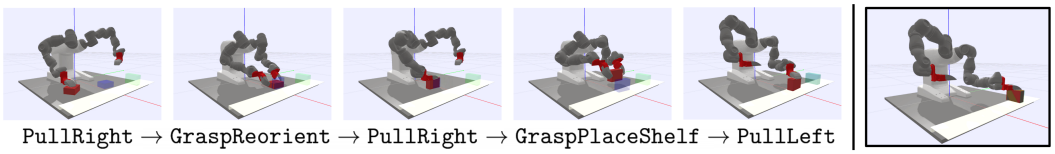


Figure 6: We show execution of a 5-step plan skeleton, where the goal configuration is on an elevated shelf. Sampling a subgoal on a shelf fits directly in our framework, through the use of our segmentation mask-based reorientation subgoal representation.

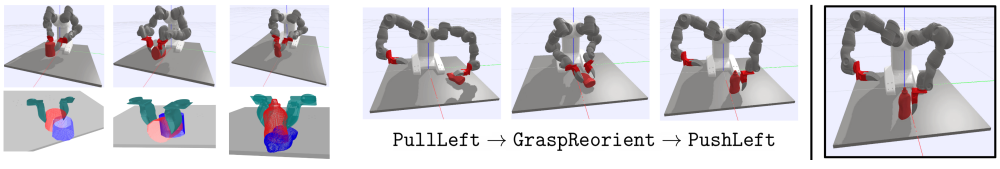
trials where a plan is found. 200 solvable problems are constructed for each skeleton. The results in Table 1 indicate that our learned models provide a large benefit in planning efficiency over the baseline that was hand-designed for cuboidal objects. The baseline's lower performance is primarily due to sampling many infeasible sequences, whereas our model learns to bias samples toward parameters that are likely to be feasible and lead to successful execution (see appendix for more detailed failure mode analysis and comparison). The similar pose errors indicate feasible plans obtained by both samplers are of similar quality.

4.3 Qualitative Results and Capabilities

Multiple Placement Surfaces Our mask-based subgoal representation enables sampling grasping subgoals on different surfaces, since arbitrary target point-clouds, obtained using plane segmentation, can be used during registration. We demonstrate this capability in Figure 6, where a book is moved from a flat resting position on a table to a vertical pose on an elevated shelf with a 5-step plan.

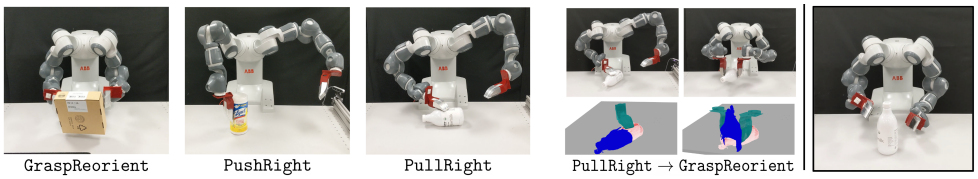
Generalization to Novel Geometry Classes Figure 7a shows *grasp-reorient* predictions trained only on cuboids, and tested on a cylindrical object and on an object in the shape of a bottle. Predictions for both T^o and T^{p_c} are quite sensible and were executed successfully in the simulator. Figure 7b shows a 3-step plan found with our planning algorithm on the bottle-shaped object. The appendix contains additional results and discussions related to generalization to novel geometric classes.

Real Robot Experiments In light of COVID-19, access to robots in our lab has been limited. However, as the primitives we use in this work were originally designed and validated with real robot experiments [9], we focus here on qualitatively demonstrating the capabilities of the framework on a real ABB YuMi robot using real perceptual inputs. Figure 8a shows snapshots from skills being executed on a variety of objects, after obtaining contact and subgoal parameters from the learned samplers (trained only in simulation) using real point-clouds as input. Figure 8b shows a 2-step plan obtained by the full framework being executed with a bottle. We used calibrated Intel RealSense D415



(a) (b)

Figure 7: (a) Predictions made from point-clouds of objects of novel geometric classes, by our model which was only trained on cuboids. (b) Executing a plan found for a 3-step plan skeleton on a bottle-shaped object



(a) (b)

Figure 8: (a) Execution of skills using parameters obtained from our learned samplers with real point-clouds as input. (b) 2-step plan executed on a real bottle-shaped object.

RGB-D cameras at the table corners to obtain point-clouds, AR tags to specify T^o_{des} for multi-step planning, and a pretrained neural network [26, 27] combined with point-cloud cropping heuristics for segmentation. Finally, we conducted additional simulated experiments (as in Section 4.2) using noise-corrupted point-clouds. The results showed a negligible performance change, further validating the performance under realistic sensing conditions (see appendix for details).

5 Related Work

Perception-based Multi-step Manipulation Most similar to this work is Fang et al. [23], that decouples subgoals and robot actions, and learns samplers to enable multistep planning from segmented object pointclouds. In contrast to us, they learn samplers for forward simulation both in the subgoal and the action space. We bypass the problem of learning low-level dynamics in the action space and use $SE(3)$ transformations to forward propagate subgoals. Moreover, they only considered planar tasks with a straight-line push primitive. In [28], a pixel-wise Q-function represents contact locations and sequencing behavior between pushing and grasping is obtained using reinforcement learning.

Task-Oriented Contact-Affordances In [29], a DexNet [30] grasp detector is extended to use a task-specific reward to select grasps that are useful for a downstream task. In [7], keypoint affordances are used for task specification, grasping, and planning motions to manipulate categories of objects. [22] extends this approach by learning the keypoints based on the task rather than manually specifying them. [8] combines learned push affordances with a mechanics model to rank push locations based on predicted outcomes and the task at hand. We approach predicting task-oriented contacts by modeling the joint distribution over reachable subgoals and corresponding contacts for a set of primitive skills.

Manipulation with 3D Deep Learning Mousavian et al. [20] and Murali et al. [21] train a VAE [31] based on PointNet++ [19] to generate and refine feasible grasps given partial point-clouds. In [32], the authors propose a scene dynamics model trained to predict $SE(3)$ transformations of point-clouds, but their model operates over low-level actions rather than high-level skills, limiting the object manipulation demonstrations to short-horizon interactions.

Learning to guide TAMP Several algorithms exist for learning to guide TAMP planners [33, 34, 35, 36, 37]. In [33], the authors consider a similar problem setup as ours in which the plan skeleton is given, and the goal is to find the continuous parameters of the skeleton. Instead of learning samplers for each manipulation skill, they learn to predict the parameters for the entire skeleton. [34, 35, 36], like us, consider learning a sampler for each manipulation skill to improve planning efficiency. Unlike our setup, however, these methods assume that the poses and shapes of objects are perfectly estimated. Our framework can be seen as extending these works to problems in which the robot must reason about objects with unknown shapes and poses by directly learning to map a sensory observation to a distribution over promising parameters of manipulation skills.

6 Conclusion

This paper presents a method to enable multistep sampling-based planning using primitive manipulation skills, when provided with segmented environment point-clouds. Our approach uses deep conditional generative modeling to map point-cloud observations to a distribution over likely contact poses and subgoals, which are used as inputs to the skills. Novel technical aspects of our approach enable the ability to use the skills effectively, and our learned samplers provide planning efficiency gains over a manually designed baseline sampler. Our qualitative results validate that the approach transfers to real point-clouds, can scale toward enabling manipulating more complex object geometry, and is compatible with solving more sophisticated multistep tasks.

Acknowledgments

This work was supported in part by the DARPA Machine Common Sense Grant and the Amazon Research Awards. Anthony and Yilun are supported in part by NSF Graduate Research Fellowships. We would like to thank the anonymous reviewers for their helpful comments and feedback.

References

[1] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *IEEE Conference on Robotics and Automation*, 2011.

[2] S. Cambon, R. Alami, and F. Gravot. A hybrid approach to intricate motion, manipulation, and task planning. *International Journal of Robotics Research*, 2009.

[3] C. R. Garrett, T. Lozano-Peréz, and L. P. Kaelbling. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *International Journal of Robotics Research*, 2014.

[4] M. Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. *International Joint Conference on Artificial Intelligence*, 2015.

[5] T. Hermans, J. M. Rehg, and A. Bobick. Affordance prediction via learned object attributes. In *IEEE International Conference on Robotics and Automation : Workshop on Semantic Perception, Mapping, and Exploration*, pages 181–184. Citeseer, 2011.

[6] A. Zeng, S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, et al. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *2018 IEEE international conference on robotics and automation*, pages 1–8. IEEE, 2018.

[7] L. Manuelli, W. Gao, P. Florence, and R. Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. *arXiv preprint arXiv:1903.06684*, 2019.

[8] A. Kloss, M. Bauza, J. Wu, J. B. Tenenbaum, A. Rodriguez, and J. Bohg. Accurate vision-based manipulation through contact reasoning. *arXiv preprint arXiv:1911.03112*, 2019.

[9] F. R. Hogan, J. Ballester, S. Dong, and A. Rodriguez. Tactile dexterity: Manipulation primitives with tactile feedback, 2020.

[10] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in neural information processing systems*, pages 5074–5082, 2016.

[11] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation*, pages 2786–2793. IEEE, 2017.

[12] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

[13] N. Chavan-Dafle, R. Holladay, and A. Rodriguez. Planar in-hand manipulation via motion cones. *The International Journal of Robotics Research*, 39(2-3):163–182, 2020.

[14] T. Lozano-Pérez and L. P. Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3684–3691. IEEE, 2014.

[15] D. Driess, J.-S. Ha, and M. Toussaint. Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image. *arXiv preprint arXiv:2006.05398*, 2020.

[16] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling. Sample-based methods for factored task and motion planning. In *Robotics: Science and Systems (RSS)*, 2017. URL http://lis.csail.mit.edu/pubs/garrett-rss17.pdf.

[17] B. Kim and L. Shimanuki. Learning value functions with relational state representations for guiding task-and-motion planning. *Conference on Robot Learning (CoRL)*, 2019.

[18] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. In *Advances in neural information processing systems*, pages 3483–3491, 2015.

[19] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[20] A. Mousavian, C. Eppner, and D. Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2901–2910, 2019.

[21] A. Murali, A. Mousavian, C. Eppner, C. Paxton, and D. Fox. 6-dof grasping for target-driven object manipulation in clutter. *arXiv preprint arXiv:1912.03628*, 2019.

[22] Z. Qin, K. Fang, Y. Zhu, L. Fei-Fei, and S. Savarese. Keto: Learning keypoint representations for tool manipulation. *arXiv preprint arXiv:1910.11977*, 2019.

[23] K. Fang, Y. Zhu, A. Garg, S. Savarese, and L. Fei-Fei. Dynamics learning with cascaded variational inference for multi-step manipulation. *Conference on Robot Learning (CoRL)*, 2019.

[24] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. *GitHub repository*, 2016.

[25] T. Chen, A. Simeonov, and P. Agrawal. AIRobot. https://github.com/Improbable-AI/airobot, 2019.

[26] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[27] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[28] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE, 2018.

[29] K. Fang, Y. Zhu, A. Garg, A. Kurenkov, V. Mehta, L. Fei-Fei, and S. Savarese. Learning task-oriented grasping for tool manipulation from simulated self-supervision. *The International Journal of Robotics Research*, 39(2-3):202–216, 2020.

[30] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[31] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[32] A. Byravan and D. Fox. Se3-nets: Learning rigid body motion using deep neural networks. In *2017 IEEE International Conference on Robotics and Automation*, pages 173–180. IEEE, 2017.

[33] B. Kim, L. P. Kaelbling, and T. Lozano-Pérez. Learning to guide task and motion planning using score-space representation. In *IEEE International Conference on Robotics and Automation*, 2017.

[34] B. Kim, L. Pack Kaelbling, and T. Lozano-Perez. Guiding search in continuous state-action spaces by learning an action sampler from off-target search experience. In *Proceedings of the 32th AAAI Conference on Artificial Intelligence (AAAI). To appear*. AAAI Press, 2018. URL http://lis.csail.mit.edu/pubs/kim-aaai18.pdf.

[35] B. Kim, L. P. Kaelbling, and T. Lozano-Perez. Adversarial actor-critic method for task and motion planning problems using planning experience. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019. URL http://lis.csail.mit.edu/pubs/kim-aaai19.pdf.

[36] T. Rohan Chitnis, Lozano-Pérez and L. P. Kaelbling. Learning quickly to plan quickly using modular meta-learning. In *IEEE International Conference on Robotics and Automation*, 2019.

[37] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel. Guided search for task and motion plans using learned heuristics. In *2016 IEEE International Conference on Robotics and Automation*, pages 447–454. IEEE, 2016.

[38] J. Paul. Besl and neil mckay, a methode for registration of 3d shapes. *IEEE Transactions on pattern analysis and machine intelligence*, 14(2):239–256, 1992.

[39] Q.-Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[40] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[41] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[42] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[43] J. J. Kuffner. Effective sampling and distance metrics for 3d rigid body path planning. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 4, pages 3993–3998. IEEE, 2004.

[44] D. Q. Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009.

[45] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[46] M. S. Ahn, H. Chae, D. Noh, H. Nam, and D. Hong. Analysis and noise modeling of the intel realsense d435 for mobile robots. In *2019 16th International Conference on Ubiquitous Robots (UR)*, pages 707–711. IEEE, 2019.

[47] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

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

[49] I. A. Sucan and S. Chitta. MoveIt. https://moveit.ros.org/.

[50] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[51] M. Van der Merwe, Q. Lu, B. Sundaralingam, M. Matak, and T. Hermans. Learning continuous 3d reconstructions for geometrically aware grasping. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11516–11522. IEEE, 2020.

A.1 Appendix

A.1.1 Point-cloud Registration

For the mask-based representation, we use Iterative Closest Point (ICP) [38] to solve for the registration and obtain the subgoal transformation. The segmented table point-cloud is used as the target. The initial transformation is combined from two sources: (1) the planar translational component of the neural network's direct subgoal prediction, and (2) a pure forward $\frac{\pi}{2}$ pitch about the object body frame. We use the Open3D [39] implementation of point-to-point ICP, which we find to work effectively. Note that other registration techniques and methods for obtaining an initial transformation could equivalently be used in our framework.

Subgoal predictions for the *pull* and *push* skills, which only operate in $SE(2)$, do not suffer from the same physical inaccuracy issues as *grasp-reorient*. Therefore, subgoals for these skills are sampled by projecting T^o predicted from the model to $SE(2)$.

A.1.2 Sampling-based Planning Algorithm

Algorithm 1 describes our sampling-based planning algorithm in detail. Nodes in a search tree are initialized with InitPointCloudNode. Assuming some initial point-cloud and samples for the skill subgoal and contact pose, a node is specified based on the subgoal and contact pose, the resulting point-cloud after being transformed by the subgoal transformation, and the parent node whose point-cloud was used to sample the corresponding parameters.

Buffers for each step in the plan skeleton are used to store nodes containing parameters that are found to be feasible. When sampling at each step, a node is randomly popped from the buffer for the corresponding step t and the point-cloud from that node is used as input for the skill sampler. Parent nodes are specified based on their index in their respective buffer.

The SatisfiesPreconditions procedure operates on the point-cloud from the popped node to determine if further subgoal/contact sampling will proceed. These precondition checks improve efficiency by avoiding sampling many point-cloud configurations that are likely or guaranteed to fail, by leveraging reachability constraints that are easy to check and specify with respect to the point-cloud. For instance, for *grasp-reorient*, we only accept point-clouds whose average planar positional coordinates are within a valid rectangular region near the front-center of the table. A similar precondition for *pull* and *push* checks if the average planar point-cloud position is within the boundaries of the table. We use the same preconditions sampler variants.

The FeasibleMotion procedure checks if the motion corresponding to the sampled parameters is feasible. This process includes two subprocedures: (1) checking for collisions between the robot and the environment at the initial and final configurations of the skill, (2) checking for collisions and kinematic feasibility of the motion computed by the low-level primitive planners. Both subprocedures must pass for the skill to be feasible. We only run subprocedure (2) if subprocedure (1) passes, as (1) is computationally cheaper and helps filter out infeasible motions more efficiently. At each step in PS, the skill samplers have a maximum number of samples, K_{max}, to obtain a set of feasible parameters. If they fail to do so within this limit, sampling moves on to the next step in the skeleton. On the final step T, the required transformation is directly computed based on the sequence of subgoal transformations that leads to the node that has been popped, starting from the root node. If the final subgoal transformation is not feasible, sampling begins all over again at the first step in PS, so that new start point-clouds continue to be added to the buffers. In our experiments we used $K_{max} = 10$.

When a feasible set of parameters is found on the final step, the ReturnPlan procedure is used to backtrack from the final node to the root node along the parents (similarly as in RRT [40]) and return the plan to be executed. If a feasible plan is not found within 5 minutes, the planner returns a failure.

A.1.3 Conditional Variational Auto-encoder Skill Samplers

The CVAE is trained to maximize the evidence lower bound (ELBO) of the conditional log-likelihood of the data in D_π. We present a brief description of using the CVAE framework in our setup (for detailed derivation please see [31, 18]).

Algorithm 1 Multistep Planning

Input: Desired transformation T^o_{des}, Point-cloud X, Plan skeleton PS, Number of skeleton steps T, Buffers for each step in skeleton $\{\mathcal{B}_t\}_{t=1}^T$, Skill parameter sampling distribution for the skill at each step in skeleton $\{p_{\pi_t}(\cdot|X)\}_{t=1}^T$, Maximum number of samples at each step in the skeleton K_{max}

root_node $\leftarrow$ InitPointCloudNode(X, I^4, None, None) ▷ Initialize root node with initial point-cloud, identity transformation, no contact pose, and no parent
$\mathcal{B}_1, ..., \mathcal{B}_T \leftarrow \emptyset$ ▷ Empty skill buffers
$\mathcal{B}_1 \leftarrow \mathcal{B}_1 \cup$ root_node
done $\leftarrow$ False
while not done **do**
  for skill step t in 1,...,T **do**
    $k \leftarrow 1$ ▷ If feasible parameters not found after K_{max} samples, move to next step
    while $k < K_{max}$ **do**
      node $\sim \mathcal{B}_t(\cdot)$ ▷ Sample start state from corresponding buffer
      $X \leftarrow$ node.pointcloud
      if PreconditionsSatisfied(X) **then** ▷ Check if point-cloud is valid for sampling
        $T^o, T^{p_c} \sim p_{\pi_t}(\cdot|X)$ ▷ Use point-cloud to sample from skill model
        if t equals T **then**
          $T^o \leftarrow$ GetFinalTransformation(node)
          if FeasibleMotion(π_T, T^o, T^{p_c}) **then**
            $X_{final} \leftarrow$ TransformPointCloud(T^o, X)
            final_node $\leftarrow$ InitPointCloudNode(X_{final}, T^o, T^{p_c}, node)
            done $\leftarrow$ True
          end if
        else
          if FeasibleMotion(π_t, T^o, T^{p_c}) **then**
            $X_{new} \leftarrow$ TransformPointCloud(T^o, X)
            new_node $\leftarrow$ InitPointCloudNode(X_{new}, T^o, T^{p_c}, node)
            $\mathcal{B}_{t+1} \leftarrow \mathcal{B}_{t+1} \cup$ new_node ▷ Add to buffer for next step
          end if
        end if
      end if
      $k \leftarrow k + 1$
    end while
  end for
  if timed out **then**
    return None
  end if
end while
plan $\leftarrow$ ReturnPlan(final_node) ▷ Backtrack through parents from final node
return plan $(T^{p_c}_{1:T}, T^o_{1:T})$

Following [18], we denote *inputs* $\mathbf{x}$, *outputs* $\mathbf{y}$, and latent variables $\mathbf{z}$. We wish to model the distribution $p_\theta(\mathbf{y}|\mathbf{x})$ (this is what is used as the skill sampler during planning). We can do so by considering a latent variable model, $p_\theta(\mathbf{y}|\mathbf{x}) = \int p_\theta(\mathbf{y}|\mathbf{x},\mathbf{z})p_\theta(\mathbf{z})d\mathbf{z}$. In doing so we assume our data is generated via the following generative process: given observation $\mathbf{x}$, latent variable $\mathbf{z}$ is drawn from prior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ and output $\mathbf{y}$ is generated from the conditional distribution $p_\theta(\mathbf{y}|\mathbf{x},\mathbf{z})$. The training procedure for the CVAE uses neural networks to approximate the generative distribution $p_\theta(\mathbf{y}|\mathbf{x},\mathbf{z})$ along with a recognition model $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$. $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$ and $p_\theta(\mathbf{y}|\mathbf{x},\mathbf{z})$ can be interpreted as a probabilistic encoder and decoder, respectively. The ELBO of the conditional log-likelihood,

$$\log p_\theta(\mathbf{y}|\mathbf{x}) \geq -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})||p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}[\log p_\theta(\mathbf{y}|\mathbf{x},\mathbf{z})] \tag{2}$$

is used as a surrogate objective function to optimize with respect to parameters θ and ϕ. In our setup we assume $p_\theta(\mathbf{z}) = p_\theta(\mathbf{z}|\mathbf{x})$.

To optimize the ELBO, the encoder $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$ is trained to map the data to a latent conditional distribution, which is constrained by a KL-divergence loss to resemble a prior $p_\theta(\mathbf{z})$ (in our case, a unit Gaussian), while the decoder $p_\theta(\mathbf{y}|\mathbf{x},\mathbf{z})$ is trained to use samples from the latent space to reconstruct the data. An estimator of the ELBO is used as the loss to optimize when training the neural networks. The estimator enables taking gradients with respect to the network parameters, and it is obtained using the reparameterization trick,

$$\mathcal{L}_{\text{CVAE}}(\mathbf{x},\mathbf{y};\theta,\phi) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})||p_\theta(\mathbf{z})) + \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\mathbf{y}|\mathbf{x},\mathbf{z}^{(l)}) \tag{3}$$

$$\mathbf{z}^{(l)} = g_\phi(\mathbf{x},\mathbf{y},\epsilon^{(l)}),\ \epsilon^{(l)} \sim \mathcal{N}(\mathbf{0},\mathbf{I}) \tag{4}$$

We can use the closed-form expression for the KL divergence between the prior $p_\theta(\mathbf{z})$ and the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$ when both distributions are assumed to be Gaussian [31]. The expression uses the mean μ and standard deviation σ of the approximate posterior, where μ and σ are outputs predicted by the encoder network.

Joint Skill Sampler Models: In our joint models, observations $\mathbf{x}$ are the point-cloud observation X, and output variables $\mathbf{y}$ include T^o, T^{p_c}, and point-cloud segmentation mask X_{mask}. The latent variable model in this setup is

$$p(T^o, T^{p_c}, X_{mask}|X) = \int p(T^o, T^{p_c}, X_{mask}|X,\mathbf{z})p(\mathbf{z})d\mathbf{z}$$

Independent Skill Sampler Models: In the baselines that use independent models, output variables $\mathbf{y}$ are separated into two independent sets, with corresponding independent latent variables. $\mathbf{y_1}$ includes T^o and X_{mask}, while $\mathbf{y_2}$ includes T^{p_c}. We represent these as two separate CVAEs:

$$p(T^o, T^{p_c}, X_{mask}|X) = p(T^o, X_{mask}|X)p(T^{p_c}|X) \tag{5}$$

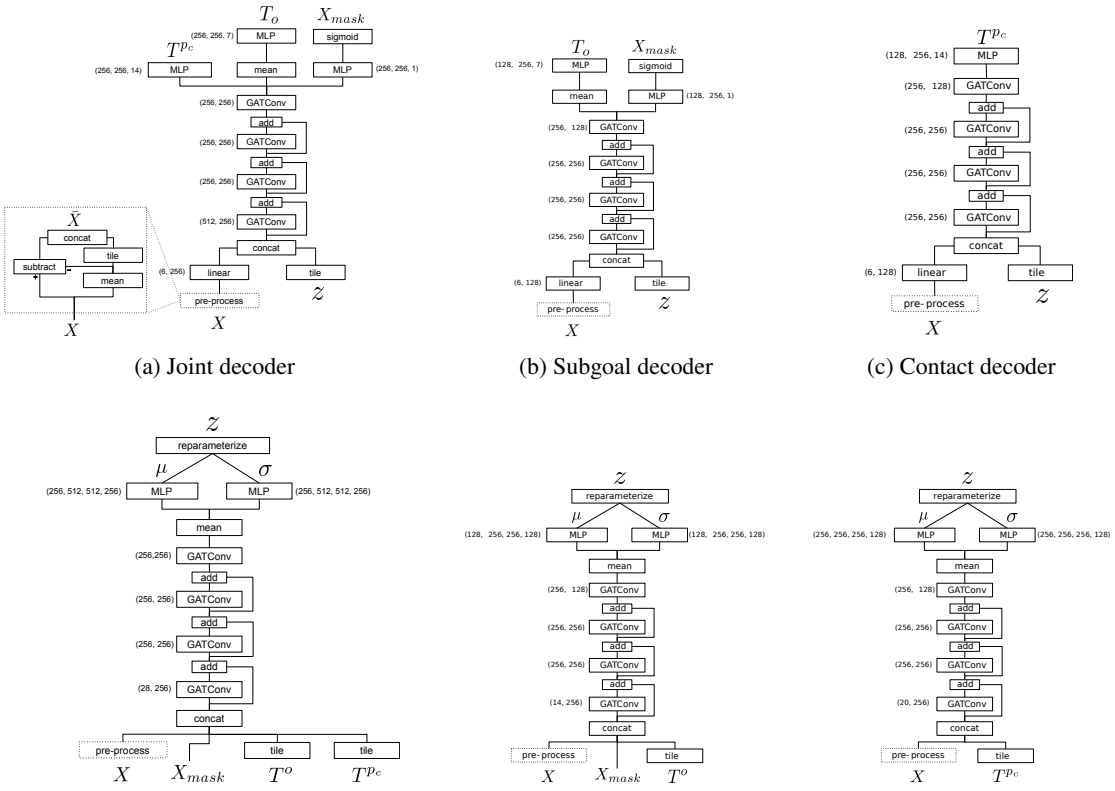
$$p(T^o, X_{mask}|X) = \int p(T^o, X_{mask}|X,\mathbf{z_1})p(\mathbf{z_1})d\mathbf{z_1} \tag{6}$$

$$p(T^{p_c}|X) = \int p(T^{p_c}|X,\mathbf{z_2})p(\mathbf{z_2})d\mathbf{z_2} \tag{7}$$

A.1.4 Neural Network Architecture Details

Inputs and Outputs Subgoals T^o and contact poses T^{p_c} are represented as 7-dimensional vectors, made up of a 3D position for the translational component, and a unit quaternion for the rotational component. Unit quaternions are directly regressed as an unnormalized 4-dimensional vector, which is normalized after prediction.

During our experiments, we found overall training and evaluation performance is improved when the decoder is trained to predict X_{mask} and T^o together (i.e. as complementary auxiliary tasks), even if T^o is not used for executing the skill. Based on this observation and to simplify implementation, all decoder models that are used to predict subgoals were set up in this way to predict T^o and X_{mask} jointly.



(a) Joint decoder (b) Subgoal decoder (c) Contact decoder

(d) Joint encoder (e) Subgoal encoder (f) Contact encoder

Figure A1: Network architectures based on Graph Attention Networks [41] for joint (a,d) and independent (b-c,e-f) CVAE decoder and encoder. The pre-processing block for all architectures is identical to the one shown in (a). GATConv layers perform self-attention over nodes in a neighborhood to propagate node features (see [41] for details). We represent the input point-cloud as a fully connected graph.

Before being input to any of the neural networks, the point-cloud X,

$$X = \begin{bmatrix} \mathbf{p_1} \\ ... \\ \mathbf{p_N} \end{bmatrix} = \begin{bmatrix} x_i, y_i, z_i \\ ... \\ x_N, y_N, z_N \end{bmatrix}$$

is pre-processed with the following steps: X is converted into $\bar{X}$, by computing the mean over all the points $X_c = \sum_{i=1}^{N} \mathbf{p_i} = [x_c, y_c, z_c]$, subtracting X_c from all the points in X, and concatenating X_c as an additional feature to all the zero-mean points to obtain

$$\bar{X} \in \mathbb{R}^{N \times 6} = \begin{bmatrix} \mathbf{p_1} - X_c \oplus X_c \\ ... \\ \mathbf{p_N} - X_c \oplus X_c \end{bmatrix}$$

We uniformly downsample the observed point-cloud to a fixed number of points before converting to $\bar{X}$ and computing the network forward pass. We use $N = 100$ points in all experiments with learned models. For the baseline samplers we use the full dense point-cloud, which can have a variable number of points.

Model Architecture based on Graph Attention Networks Figure A1 shows the GAT-based architectures for the CVAE encoder and decoder. The encoder is a Graph Attention Network, which is trained to represent the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y})$. To combine the input variables $\mathbf{x}$ ($\bar{X}$), and the output variables $\mathbf{y}$, we concatenate all the variables in $\mathbf{y}$ as extra point-wise features to $\bar{X}$. T^o and T^{p_c} are repeated and concatenated with all the points in $\bar{X}$, and the point-wise binary features in $X_{mask} \in \mathbb{R}^N$ are directly concatenated with the corresponding points in $\bar{X}$.

Output point features are computed using the graph-attention layers in the encoder. A mean is taken across all the points to obtain a global feature encoding, which is then provided to separate

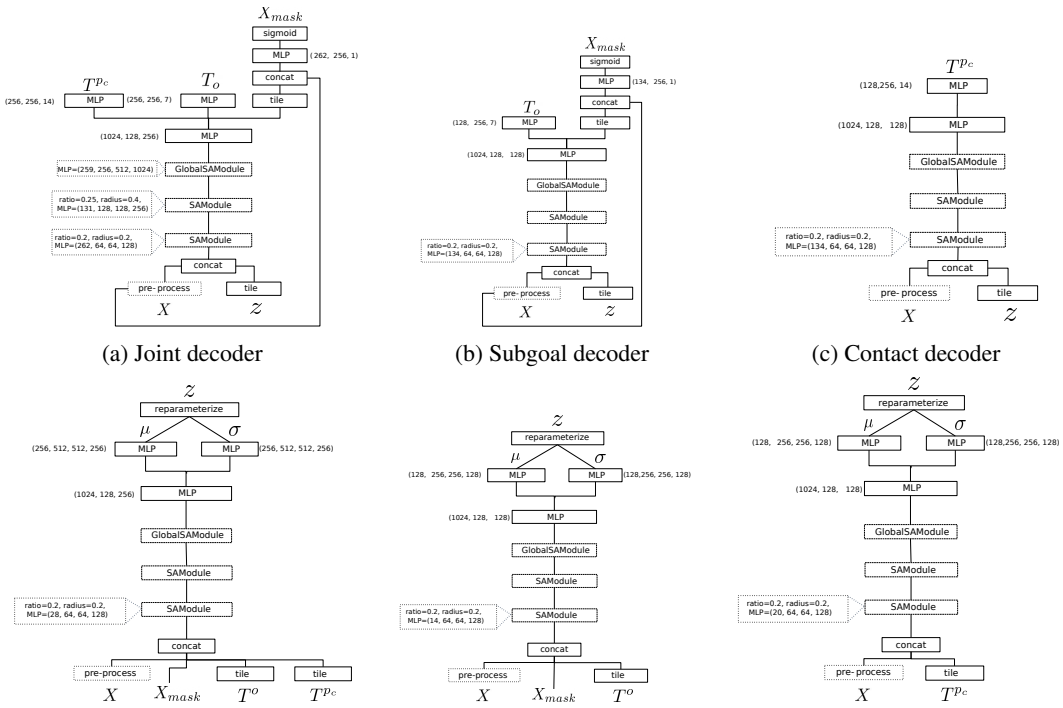


Figure A2: Network architectures based on PointNet++ [19] for joint (a,d) and independent (b-c,e-f) CVAE decoder and encoder. SAModules perform the hierarchical set abstraction operation in the PointNet++ (for details see [19]). The only difference between the initial SAModules in the various models is the input dimension.

fully-connected output heads that compute μ and σ. μ and σ are reparameterized to compute $\mathbf{z}$, and are used to compute the KL divergence loss [31] in $\mathcal{L}_{\text{CVAE}}$.

The decoder receives as input $\bar{X}$ and $\mathbf{z}$. $\bar{X}$ is projected to a higher dimensional space with a learnable weight matrix W. $\mathbf{z}$ is repeated and concatenated with each point in $W\bar{X}$ as an additional per-point feature. This representation is then provided to another Graph Attention Network to obtain output point features.

The point features are used by different output heads to reconstruct the data. *Each* point feature is independently used by a fully-connected output head to predict the contact pose, and another fully-connected output head followed by a sigmoid to predict the binary mask. In parallel, the point features are averaged and passed to a third fully-connected head that predicts T^o.

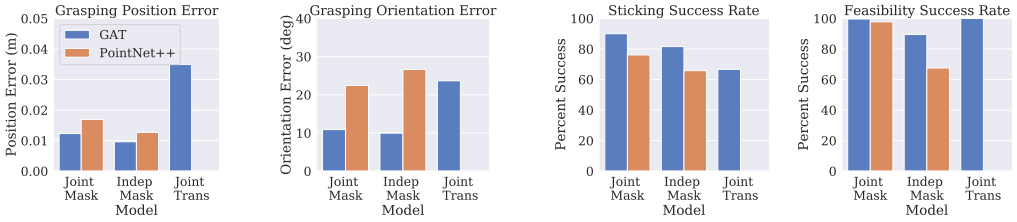
Model Architecture based on PointNet++ The encoder and decoder have a PointNet++ architecture [19], shown in Figure A2. X is converted to $\bar{X}$ and concatenated with all the $\mathbf{y}$ variables before being passed to the encoder, in the same way as described above . The PointNet++ encoder computes a global point-cloud feature, and fully-connected output heads predict μ and σ which are reparameterized as $\mathbf{z}$.

During decoding, X converted to $\bar{X}$, $\mathbf{z}$ is concatenated as a per-point feature with $\bar{X}$, and a global point-cloud feature is computed with a PointNet++ decoder. Separate fully-connected heads use the global feature to make a single contact pose prediction for T^{p_c} and transformation T^o. The global feature is concatenated to all the original points in $\bar{X}$ and passed to a third fully-connected output that predicts binary point labels for X_{mask}.

CVAE Inference During testing, a latent vector $\mathbf{z}$ is sampled from the prior $p_\theta(\mathbf{z})$ and the trained decoder follows the same respective processes as described above with the trained [GAT or PointNet++] layers and output heads. We find it worked well to randomly sample one of the per-point T^{p_c} predictions for use in the executed skill when using the GAT-based model. In our experiments

Table 2: Training data statistics

Skill Type	Number of objects	Number of samples
Pull	110	12712
Grasp-Reorient	64	8408
Push	223	3835



(a) (b) (c) (d)

Figure A3: More detailed single-step ablation results for *grasp-reorient* skill. (a) and (b) show the average position (a) and orientation (b) error between the subgoal transformation provided to the skill and the actual transformation the object underwent between its initial configuration and final configuration after the skill was executed. (c) shows the fraction of trials where a contact was maintained with the object through the duration of the reorientation and (d) shows the fraction of trials where a feasible motion was found within 15 samples from the learned model.

we used latent vectors of dimension 256 for the joint models and 128 for independent contact and subgoal models.

A.1.5 Training Data Generation and Training Details

We generate training data by simulating the primitive skills with the known 3D models of the cuboids, sampling transitions between stable object poses for subgoals, and sampling the object mesh for contact poses. If the skill is feasible and successfully moves the object to the subgoal, the parameters are added to the dataset. We also record the points in the point-cloud that end within Euclidean distance thresholded from the table as ground truth labels for X_{mask}.

The number of objects and dataset size used for training each skill sampler are shown in Table 2. Because the data generation process has a high degree of stochasticity and the amount of time required to obtain a large number of successful samples varies between skills, the number of data point/object used for each skill was not consistent.

During training, we use the Adam optimizer [42] with learning rate 0.001 to minimize the $\mathcal{L}_{\text{CVAE}}$ loss function with respect to the parameters of the CVAE encoder and decoder. The reconstruction term in the loss is broken up into separate components: We use an MSE loss for the positional components of T^o and T^{p_c}, and the below geodesic orientation loss [43, 44],

$$\mathcal{L}_{\text{orientation}} = 1 - \langle q_{\text{predicted}}, q_{\text{ground-truth}} \rangle^2 \tag{8}$$

for the orientation components in T^o and T^{p_c}, where q denotes the 4-dimensional quaternion components of the subgoal transformation/contact pose and $\langle\ ,\ \rangle$ denotes an inner product. We use a binary-cross entropy loss for X_{mask}. For the GAT-based models that make N separate predictions of T^{p_c}, the reconstruction loss is computed separately for all of the predictions with respect to the single ground truth training example.

A.1.6 Evaluation Details

Global Success Rate Breakdown The global success rate used in our single-step skill ablations is designed to aggregate multiple performance factors into a single metric. Here we break these results down to examine the components that lead to the overall trends shown in Figure 4. This section focuses on the *grasp-reorient* skill, as it is most influenced by our particular design choices (similar trends were observed for *push*, while each ablated variation technique worked similarly well for learning the *pull* skill).

Figures A3a and A3b show the average position and orientation errors over all the single-step trials for variants of the learned samplers. Figures A3c and A3d shows success rate based on whether contact was maintained during skill execution (c) and whether a feasible motion was found within 15 samples from the neural network (d).

From the larger position and orientation errors, and reduced sticking success rate, we can see the significant benefit provided by the mask-based subgoal representation (Joint/Indep Mask) over directly using the transformation predicted by the model (Joint Trans). The large position and orientation errors for Joint Trans are typically due to the predicted transformation moving the object to a position vertically above the table. In these cases the robot moves the object above the table, breaks contact, and the object passively falls onto the table. The large errors in Figure A3a and A3b reflect the difference between the object's "in-air" pose and wherever it ends up on the table after the robot breaks contact. Directly predicting T^o with the Joint Trans architecture also leads to point-cloud configurations that penetrate the table. This frequently causes the robot to lose contact when trying to reach these subgoals, because an unexpected contact between the table and the object occurs before skill execution is complete.

The performance between the PointNet++-based and the GAT-based models are supported by the large difference in orientation error, shown in Figure A3b. This is due to the end-effector/object alignment issues depicted in Figure 5a-b, which cause an unexpected object rotation during the initial grasp. We also observe the PointNet++ encoder reduces success rates for sticking and feasibility in Figures A3c and A3d. This is because many contact poses predicted by the PointNet++ model lead to collision with the environment, and are more likely to lose contact with the object.

Finally, we see in Figure A3d that predicting the contact and subgoal parameters jointly (Joint Mask/Trans) instead of independently (Indep Mask) leads to improved Feasibility Success Rate. This is because the Indep models require many more samples to find contact poses and subgoals that work well together. In contrast, the shared representation learned by the Joint models better captures this dependence and helps to more efficiently find a set of parameters that are compatible with each other.

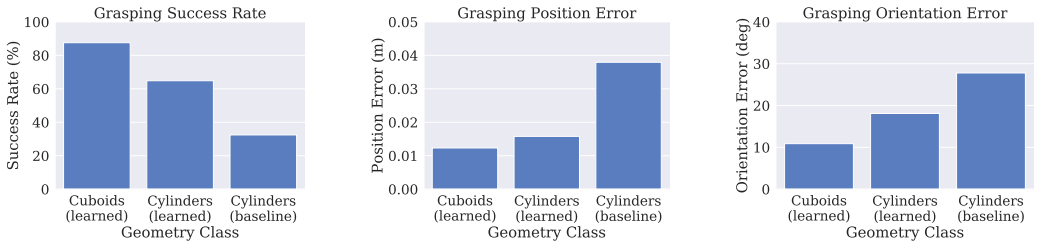
Baseline Uniform Sampler Heuristics The baselines we compare our learned samplers to in the multi-step evaluation are based on a uniform sampler that uses the point-cloud to sample potential T^o and T^{p_c} values for the *grasp-reorient* and *pull* skills. The samplers are based on the following heuristics: (1) During a *pull*, the palm should contact the object face down at a point that lies on top of the object, (2) During a *grasp-reorient* the palms should face each other and contact antipodal points that are on the side of the object, (3) During a *grasp-reorient* the subgoal T^o should encode a rigid transformation between two stable configurations of the object, that each have different faces contacting the table, (4) the robot should not contact the face of the object that will contact the table in the subgoal configuration.

For the *grasp-reorient* sampler, we first estimate the point-cloud normals and segment all the planes using the Point Cloud Library (PCL) [45]. This process is done once to avoid recomputation in each new sampled configuration; the normals and planes are all transformed together with the overall point cloud during planning. A plane is sampled and used to solve for T^o using the same registration process described in Section A.1.1. Before sampling contact poses, we then filter out points in the point-cloud that are likely to cause infeasibility if contacts were to occur at their location. Specifically, we remove the plane that is used for subgoal registration and the plane opposite to it, since contacting the object on these planes is sure to cause a collision with the table in the subgoal configuration. We also remove points below a z threshold so that the palms are less likely to contact the table in the start configuration. T^{p_c} values are determined from the points that remain after filtering. We sample antipodal points in the remaining object point-cloud for the positional component. For the orientation component, we align the palm-plane normal with the estimated normal at the sampled point, and then sample a random angle within the plane by which to rotate the pam. We only consider a range of within-plane palm angles that will not lead the wrist to collide with the table (i.e. the direction of the front of the palm must have negative z component).

For the *pull* sampler, we first estimate the point-cloud normals using PCL, and then search for a point that has an estimated normal that is close to being aligned with the positive z-axis. This point is used as the position component in T^{p_c}. The orientation component is determined by aligning the palm-plane with the positive z-axis in the world frame, and then sampling a random angle within the plane by which to rotate the palm.

Table 3: Multistep planning results on noise-corrupted point-clouds using learned samplers trained on point-clouds without noise, similar to Table 1. The depth camera noise model was obtained from [46]

Skeleton	Success Rate (%)	Pose Error (cm / deg)	Time (s)
PG	94.5 ± 3.3	1.1 ± 0.5 / 5.8 ± 3.0	30.0 ± 5.3
GP	95.9 ± 2.8	3.9 ± 0.5 / 30.1 ± 5.7	22.8 ± 5.3
PGP	87.9 ± 4.5	2.1 ± 0.3 / 16.6 ± 3.7	55.5 ± 10.4



(a) (b) (c)

Figure A4: Single-step *grasp-reorient* results on novel cylindrical objects

A.1.7 Additional Quantitative Results

Multi-step Planning Evaluation with Noisy Point-clouds To further understand the generalization capabilities of the system, we conducted single-step experiments on 10 novel cylindrical objects with 10 start poses each, using both our learned model and the hand-designed baseline sampler. For the baseline, we approximate the cylinder with an oriented bounding box. We report the global success rate from Figure 4 and the average position/orientation errors as in Figure A3, with corresponding values from the best performing model tested on cuboids for reference. Figure A4 shows the results. As expected, we see a drop in performance (larger pose errors and smaller success rate), due to the significant distributional shift in the point cloud observations. However, many of the skill executions are still successful and performance is still higher than using the baseline sampler with cylinders approximated as cuboids. We hypothesize that the performance gap can be made up by training on a more diverse set of objects.

Single-step Evaluation with Novel Geometric Classes To understand how performance differs under more realistic sensor noise, we evaluate our learned models in the multi-step setting described in Section 4.2 with noisy point-clouds. We use a depth camera noise model taken from [46] to simulate depth image noise which shows up as noise in the point-cloud that is provided to the samplers. Results shown in Table 3 demonstrate that our system is quite robust to moderate levels of input noise.

A.1.8 System Implementation Details

Model Implementation We used PyTorch [47] for training and deploying the neural network components of the framework, and we used the PyTorch Geometric library [48] for implementing the Graph Attention Network and PointNet++ layers in our samplers.

Feasibility Checks We use the `compute_cartesian_path` capability in the MoveIt! [49] motion planning package for the FeasibleMotion procedure in Algorithm 1. This procedure simultaneously converts the cartesian end-effector path computed by the skills into a sequence of joint configurations using RRTConnect [50], and checks to ensure the path does not cause singularities or collisions.

Contact Pose Refinement After T^{p_c} is determined by either the learned or uniform samplers, it is refined based on the dense pointcloud of the object, so that the robot is more likely to *exactly* instead of missing contact or causing significant penetration. The refinement is performed by densely searching along the 3D rays aligned with the palm-plane normal, beginning at the position component of T^{p_c}, and computing the distance to the closest point in the point-cloud at each point along this line. The point in the point-cloud with the minimum distance is then used to update the positional component of T^{p_c} (the orientation component is unchanged). This process provides a large practical improvement, and we use it for all the skills and all variants of the samplers we evaluated.

Table 4: Comparing the fraction of failed planning attempts that were due to specific failure modes between learned and hand-designed samplers. 50 planning problems were setup with P→G→P plan skeleton, and different reasons for failure were tracked. Values denote the percent of overall trials where a particular failure mode occurred (note modes may overlap, so percentages don't add to 100).

Type	Colliding Start (%)	Colliding Goal (%)	Path Infeasible (%)	Precondition (%)
Learned	3.5	13.4	44.2	62.4
Hand Designed	7.3	38.9	70.5	43.4

Table 5: Average number of sampling iterations performed during multi-step planning on a P→G→P skeleton over 50 trials, using the learned samplers and the hand-designed baseline. Total denotes the overall average over both successful and failed planning attempted, while Successes and Failures divides the average number of sampling iterations performed based on whether a feasible plan was found or not, respectively. Results highlight that both methods require similar computation (as indicated by the similar number of samples on failed attempts) and that the learned samplers are biased toward finding feasible parameters (as indicated by the lower number of samples required on successful attempts).

Type	Total	Successes	Failures
Learned	46.5	37.6	137.7
Hand Designed	113.3	70.7	130.1

A.1.9 Additional Discussion

Skill Capabilities and Geometry Generalization Implications In addition to the explicit assumptions we make in designing our framework, the system also implicitly inherits any additional assumptions made by the underlying skills that are utilized. For the skills from [9] used in this work, this includes the additional assumption of a limited set of possible types of contact interaction that can occur between the object and the robot/environment. For instance, the contact models used by the skills are not built to plan through rolling, as might occur when pulling a cylindrical object in certain ways. They also do not consider certain 3D interactions such as toppling, which can happen when contacting a tall thin object during a push skill.

This interplay of assumptions made by the primitive skills and the higher-level components provided by our framework limited our ability to demonstrate more sophisticated generalization to novel geometries. For instance, multi-step planning and execution on more diverse geometries may require primitive skills that account for more types of contact interactions. Designing such primitives was out of the current scope and is an important direction for future work, but the current framework could be similarly applied if such primitives were available.

Point-cloud Occlusion and Observability While we used point cloud obtained from simulated depth cameras in the scene, these point cloud are relatively complete and unoccluded, since we utilize multiple cameras placed at the corners of the table in the scene and don't deal with environments with clutter. Enabling long-horizon manipulation planning in more realistic scenarios where occlusions are significant and less sensors can be used is an important challenge that was beyond the scope of this work.

Furthermore, the issue of object observability raises the question of whether to deal with planning directly with a heavily occluded observation of the object or to augment the perception system with a shape completion module that predicts the geometry in the occluded regions, as is done in other manipulation systems that deal with limited observability [51]. Directly working with the partial point cloud would present numerous challenges for our system (and any manipulation planning framework). This is because our manipulation action parameters are represented explicitly with respect to points on the point cloud, since these denote locations where contact on the object can confidently be made with the robot or with the environment. We therefore hypothesize shape completion techniques to be more suitable and of great importance for extending the ideas presented in this work to scenarios where objects are more occluded.

Failure Modes To understand the performance gap between our learned samplers and the baselines we tracked the frequency of different types of failures during planning (Table 4) and the average

number of sampling iterations performed by both methods (Table 5). We found that the main issue with the baseline is that it spends more time checking feasibility of infeasible motions, due to sampling many contact/subgoals that are infeasible either for the robot, or with each other, or both. In this way, we can interpret part of the value added by our samplers as having learned the helpful biases that guide sampling toward regions that are likely to be feasible, similar to as in other works that learn biased samplers for motion planning and TAMP [37, 34]. We found that computation efficiency between both methods is similar (see Table 5).

Failure modes that lead to poor execution of the skills after parameters are sampled or multistep plans are found are primarily related to small errors in predicted palm poses or subgoal transforms that can lead to unintended outcomes. This includes predicting contact that is too soft so that the object slips, or predicting contact that penetrates the object too much, which causes large internal forces and robot joint torques along with highly unrealistic physical behavior in the simulator. Additionally, as usual with multistep scenarios, errors can accumulate between iterative predictions during planning. A common example is a small error in a subgoal prediction that leads the transformed point-cloud to be in a slightly unstable configuration that is not reflective of the configuration the object settles into when the skill is executed. This can lead the following skill to be executed poorly due to the nominal plan expecting the object to be in a different configuration than what it actually reached.

Finally, many failure modes we observe are generally related to the realities of running the obtained plans and low-level motions purely open-loop. Ideally, high-level replanning/plan refinement based on updated point-cloud locations at each step can be used to mitigate cascading error issues described above, and controllers developed in conjunction with the primitive skill planners [9] could be used to make local adjustments based on feedback from the contact interface to better enforce the sticking contact assumption to be true when a skill is executed.

Real-world/Real-robot Implementation Considerations Implementing the primitive skills and the framework proposed in this work on a real robotic platform requires certain considerations. In particular, practical difficulties can arise when implementing the plans that are obtained on a stiff position-controlled platform without any sensing to guide the skill execution. This is because skills like pulling and grasping are fundamentally designed to apply compressive forces on the object during execution, which can be problematic when dealing with near-rigid object and large position gains.

We found very large practical benefit introducing a passively compliant element at the wrist of the robot that deforms to take up forces encountered when commanding contact pose positions that slightly penetrate the object (this compliance was also modeled in the PyBullet YuMi simulation). Other ways of introducing non-stiff behavior can be similarly applied to realize the behaviors that are planned by the manipulation skills, such as hybrid position/force control or cartesian impedance control. Small errors in commanded positions can also lead the skill to miss making contact entirely. Guarded movements using aggregated wrist force/torque sensing or local contact sensing can help alleviate execution errors of this type (i.e. by moving to a nominal pose predicted by the sampler and slowly approaching the object in the direction of the palm normal until a certain force threshold is reached).