# Planning with Spatial and Temporal Abstraction from Point Clouds for Deformable Object Manipulation

**Xingyu Lin** ∗† **Carl Qi** ∗† **Yunchu Zhang** † **Zhiao Huang** ‡

**Katerina Fragkiadaki** † **Yunzhu Li** § **Chuang Gan** ¶ **David Held** †

**Abstract:** Effective planning of long-horizon deformable object manipulation requires suitable abstractions at both the spatial and temporal levels. Previous methods typically either focus on short-horizon tasks or make the strong assumption that full-state information is available. However, full states of deformable objects are often unavailable. In this paper, we propose PlAnning with Spatial and Temporal Abstraction (PASTA), which incorporates both spatial abstraction (reasoning about objects and their relations to each other) and temporal abstraction (reasoning over skills instead of low-level actions). Our framework maps high-dimension 3D point clouds into a set of latent vectors and plans skill sequences with the latent set representation. Our method can solve challenging, novel sequential deformable object manipulation tasks in the real world, which require combining multiple tool-use skills such as cutting with a knife, pushing with a pusher, and spreading dough with a roller. Additional materials can be found on our project website.[1]

**Keywords:** Long-horizon Planning, Deformable Object Manipulation

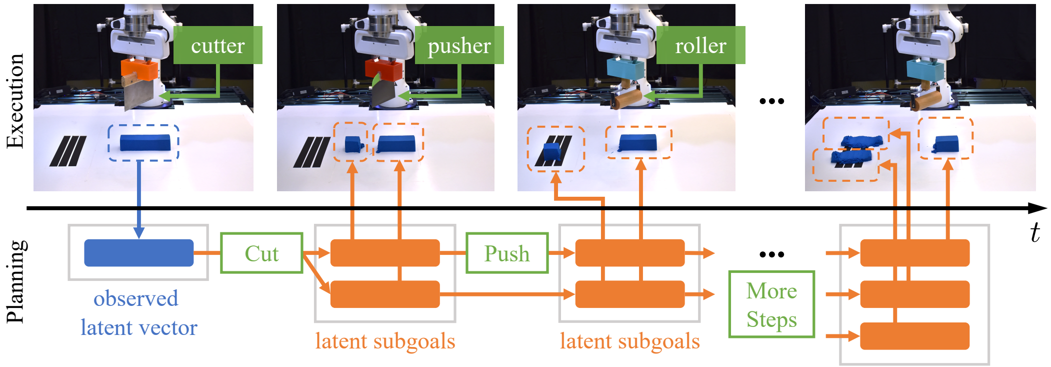


**Figure 1: Long-horizon dough manipulation with diverse tools.** Our framework is able to solve long-horizon, multi-tool, deformable object manipulation tasks that the agent has not seen during training. The illustrated task here is to cut a piece of dough into two with a cutter, transport the pieces to the spreading area on the left (with a high-friction surface) using a pusher, and then flatten both pieces with a roller.

## 1 Introduction

Consider a typical cooking task of making dumplings from dough. People plan over which piece of dough to manipulate and which tool to use in sequence, incorporating both spatial and temporal

∗ equal contribution; † CMU ‡ UC San Diego; § Stanford University; ¶ UMass Amherst & MIT-IBM Lab

[1] https://sites.google.com/view/pasta-plan

6th Conference on Robot Learning (CoRL 2022), Auckland, New Zealand.

abstractions. A spatial abstraction reasons about objects, parts, and their relations to each other, such as reasoning about pieces of dough instead of reasoning about individual dough particles; such a spatial abstraction enables efficient planning and compositional generalization. On the other hand, a temporal abstraction incorporates abstract actions represented as a set of skills. With abstract actions, the agent plans for the types and parameters of the skills to execute over a period of time, instead of making plans for low-level actions such as joint torques at each time step. Temporal abstractions allow planning at the skill level, enabling more efficient optimization for solving long-horizon tasks. An autonomous robot that operates in unstructured environments should be able to reason about world dynamics using high-level spatial and temporal abstractions instead of reasoning only over the physical state, raw sensory observation, or low-level robot actions.

The research community is making rapid progress towards developing state abstractions for manipulating deformable objects, including key points [1, 2], graphs [3, 4], dense object descriptors [5], or implicit functions [6, 7]. However, most of these approaches do not make abstractions at the temporal level, limiting their use to short-horizon tasks. Methods are also being developed with temporal abstractions, planning over a set of skills to solve long-horizon tasks [8, 9, 10]. However, the lack of spatial abstraction severely limits their generalization ability. Therefore, it remains a key question in robot learning on how to learn spatial and temporal abstractions within a unified framework for complex and long-horizon manipulation tasks.

In this work, we focus on the challenging task of sequential deformable object manipulation, as shown in Figure 1. We consider a set of dough manipulation tasks that require sequentially applying different skills using multiple tools to manipulate dough, such as spreading using a roller, cutting using a knife and pushing using a pusher, where the longest task requires applying 6 skills in sequence. Deformable objects like dough have nearly infinite degrees of freedom. As such, in this work, we dynamically cluster points in a point cloud into different groups and learn a point cloud encoder to map each element in the group into a latent vector. In this way, we obtain a compositional 3D set representation of the state space. Given an observation and a target point cloud, we then sample skill sequences with subgoals generated in this latent space. We learn skill abstraction modules to determine the feasibility and score of each skill sequence and use them for planning.

Our contribution of this paper is a framework that PlAns with Spatial and Temporal Abstraction (PASTA) by learning a set of skill abstraction modules over a 3D set representation. Our framework can compose a set of skills to solve complex tasks with more entities and longer-horizon than what was seen during training. We show that PASTA significantly outperforms an ablation that performs planning with a flat representation without a spatial abstraction (e.g. without a set representation). Finally, our planner can be trained in simulation and transferred to the real world.

## 2 Related Work

**Model-based Planning for Sequential Manipulation.** One line of research for sequential manipulation is Task and Motion Planning (TAMP). TAMP systems typically assume known object states and known effects for the action operators [11, 12, 13, 14, 15]. However, it is difficult to estimate states and dynamics for unknown objects or from partial observations. While recent works have made progress in learning certain components of the system, such as the logical states [16] from high dimensional observations or learning action models [17, 18, 19] from interactions, they still require either known states or known action operators. In contrast, we do not assume known states or action operators, and learn a 3D set representation as well as the action model with the representation.

Another approach learns dynamics directly from visual observations [3, 20, 21]. Most of these works focus on learning a one-step dynamics model for planning short-horizon tasks. A few works learn the dynamics model over a set of skills and use it for sequential manipulation of rigid objects [22, 23] or deformable objects [8]. However, these works do not use an object-centric representation and thus cannot easily generalize to more complex scenes. In contrast, our framework unifies both temporal and spatial abstraction and can perform long-horizon manipulation for complex tasks with more objects than in previous work, as we will show.

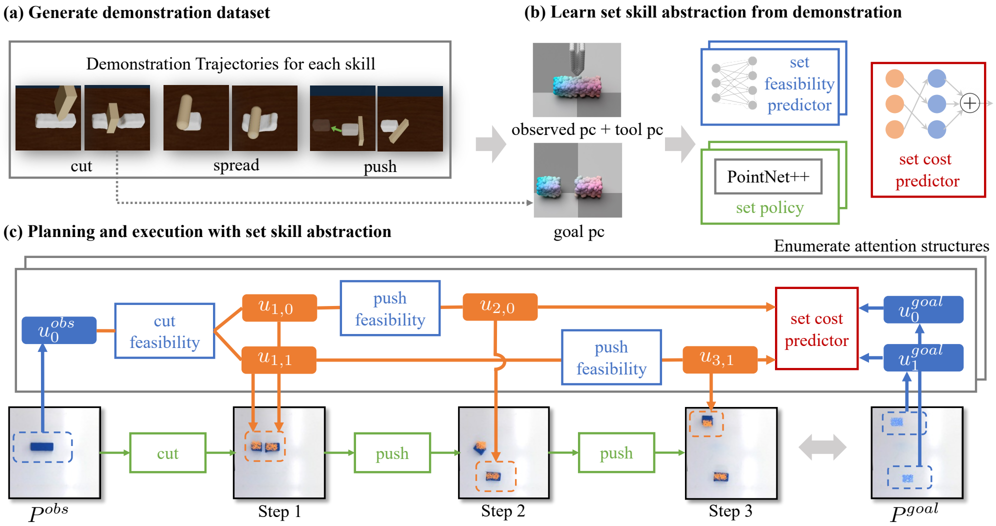


**Figure 2: Overview of our proposed framework PASTA.** (a) We first generate demonstration trajectories for each skill in a differentiable simulator using different tools. (b) We then sample point clouds (pc) from the demonstration trajectories to train our set skill abstraction modules. (c) We map point clouds into a latent set representation and plan over tool-use skills to perform long-horizon deformable object manipulation tasks. $P^{obs}$, $P^{goal}$ are the observation and target pc; $u_{i,j}$ denotes component $j$ at step $i$. The example shows our method performs the CutRearrange task, which requires cutting the dough into two pieces with a knife and transporting each piece to its target location.

**Planning with Spatial Abstraction.** Prior works leverage spatial abstraction to facilitate solving tasks that involve complex dynamics and high-dimensional observations. These works either model a compositional system with Graph Neural Networks (GNN) [3, 4, 7, 24, 25] or learn policies directly from object-centric representations [26, 27]. These works demonstrate compositional generalization, but they learn policies or one-step dynamics models for planning, which can be difficult for solving long-horizon tasks. In contrast, our framework connects temporally extended spatial abstractions with a feasibility predictor to plan over a longer time horizon. Xu et al. [28] learns a planner grounded on object-centric visual observation and can solve long-horizon tasks. However, its planner takes a symbolic goal and plans in a predefined symbolic domain to output symbolic subgoals. In contrast, our method does not require defining a symbolic planning domain.

**Deformable Object Manipulation.** Deformable objects have nearly infinite degrees of freedom and complex dynamics, making them very challenging to manipulate. Previous works have explored pouring liquid [6, 29, 30], rope manipulation [31, 32], and cloth manipulation [33, 34, 4, 35, 36]. Other papers have also explored manipulating elasto-plastic objects such as deforming them by grasping [3, 37], rolling [38, 39, 40], or cutting [41]. However, these works mostly only consider manipulation with one skill at a time. In contrast, we consider the task of sequential manipulation using multiple tools. The one exception is DiffSkill [8], where multiple skills are chained together. However, DiffSkill uses RGB-D images to represent the scene. In contrast, we use a 3D set representation that separately encodes each entity in the scene, enabling compositional generalization to tasks with more objects and longer-horizon. Furthermore, we use point clouds as the input and we are able to transfer our planner from simulation to the real world.

## 3 Method

Given a point cloud of the dough $P^{obs}$ and a goal point cloud $P^{goal}$, our objective is to execute a sequence of actions $a_1, ..., a_{T_{tot}}$ that minimizes the distance between the final observed point cloud and the goal $D(P^{obs}_{T_{tot}}, P^{goal})$ where $P^{obs}_{T_{tot}}$ is the observation point cloud at time $T_{tot}$. We aim to solve long-horizon tasks that require chaining multiple skills in novel scenes with more objects than training. To do so, we present a general framework that incorporates spatial and temporal abstractions, as summarized in Figure 2. We use point clouds as input to all our modules to enable easier transfer from simulation to the real world and to enable robustness to changes in viewpoint.

We assume access to an offline dataset of demonstration trajectories $\mathcal{D}_{demo}$ from $K$ skills, where each trajectory demonstrates one of the skills using one tool. We can learn skill policies by imitation learning from these demonstrations. To chain these skills to solve long-horizon tasks, we train a set of skill abstraction modules (Sec. ) which can be used for planning in the latent space (Sec. ).

### 3.1 Spatial Abstraction from Point Clouds

**Scene Decomposition:** First, we describe our spatial abstraction of the point cloud observation. Given a point cloud $P \in \mathbb{R}^{N \times 3}$, we first cluster the points into different components based on their proximity in space. In this paper, we apply DBSCAN [42] to $P$ and group points into a set of entity point clouds $\{P_i \in \mathbb{R}^{N_i \times 3}\}_{i=1...C}$ by separating points from high-density regions into different clusters. While other works on scene decomposition can also be used [43], we find this simple method to be sufficient for our tasks.

**Entity Encoding:** Planning directly in the high-dimensional space of point clouds is inefficient. To enable efficient planning in a latent space, we train a point cloud variational autoencoder (VAE) [44]. The VAE model includes three modules: A point cloud encoder $\phi : \mathbb{R}^{N_i \times 3} \rightarrow \mathcal{U}$ that maps each entity point cloud to a latent vector, a decoder $\psi : \mathcal{U} \rightarrow \mathbb{R}^{N_i \times 3}$ that maps from a latent space back into a point cloud, and a prior distribution over the latent space $p_u : \mathcal{U} \rightarrow [0, 1]$ which can be used to generate samples from the latent space during planning. We can then encode the point clouds $\{P_i\}$ into a set of latent vectors: $\{u_i\}_{i=1...C}$ as our set latent representation. We further achieve translation invariance by separating the translation from its shape embedding. See the Appendix for details.

### 3.2 Learning Skills by Imitation

Given the demonstration trajectories $\mathcal{D}_{demo}$ of $K$ skills, we first distill these trajectories into $K$ closed-loop policies. The input to the policy for the $k^{th}$ skill $\pi_k$ is a subset of the observed point clouds $\{P_i^o\}$ and goal point clouds $\{P_j^g\}$, and a tool point cloud $P_k^{tool}$. The policy only sees a subset of the dough point cloud and goal point cloud in the scene to enable compositional generalization to scenes with more objects. For example, a dough-spreading policy will only see the dough being spread. To achieve this, we train each set point cloud policy with behavior cloning and hindsight relabeling [45] on the demonstration dataset with an attention mask that filters out the non-relevant entity point clouds. During planning, this attention mask will be provided by the planner. The policy outputs an action at each timestep to control the tool directly.

### 3.3 Neural Spatial and Temporal Abstraction

We assume that each skill learned from the demonstration is only capable of performing a single-stage task with a single tool for a single object. To solve longer-horizon tasks, we further learn a feasibility predictor and a cost predictor. They can be used to plan subgoals that chain the skills into a sequence, such that each subgoal is feasible for the corresponding policy to reach and the final subgoal reaches a given goal. Additionally, all these modules take in the set representation $\{P_1 \dots P_C\}$ as input to achieve compositional generalization.

**Set Feasibility Predictor** Similar to DiffSkill [8], we train a feasibility predictor $f_k(U^o, U^g)$ for each skill, where $U^o = \{u_i^o\}_{i=1...N_o}, U^g = \{u_j^g\}_{j=1...N_g}$ are latent set representations of an observation and goal point cloud respectively. The feasibility predictor outputs a value in $[0, 1]$ denoting if the goal can be reached from the observation by executing the $k^{th}$ skill. In DiffSkill, the feasibility predictor uses a flat representation that takes in a single latent vector for all objects in the scene as input. However, as our skills such as cutting or spreading only need to take in a subset of the objects as input, we use the same attention method for the feasibility and assume that the feasibility predictor only takes as input a subset of the full set representation $\hat{U}^o \subseteq U^o, \hat{U}^g \subseteq U^g$, where $\hat{U}^o = \{u_i^o\}_{i=1...N_k}, \hat{U}^g = \{u_j^g\}_{j=1...M_k}$, Here, $N_k$ and $M_k$ are the number of components in the observation and goal for skill $k$. The number of components in the observation and goal can be different since the number of components can change before and after executing a skill; for example, the cut skill takes one component as observation and cuts it into two components. As another example for robot assembly [46], the number of entities increases when a piece is disassembled into parts and the number of entities decreases when the parts are assembled. In this work, we manually define the number of entities $N_k$, $M_k$ per skill. Determining which subset to attend to when executing each skill can be difficult; we make this decision during planning and defer the details to Sec. . We parameterize $f_k$ to be invariant to permutation using max-pooling layers. See Appendix for details.

We train the feasibility predictor of skill $k$ with positive examples $\hat{U}^o, \hat{U}^g$, where the goal $\hat{U}^g$ can be reached from the observation $\hat{U}^o$ within $T$ timesteps by executing skill $k$. The negative examples are goal $\hat{U}^g$ that cannot be reached from the observation $\hat{U}^o$ using skill $k$. During training, we obtain positive pairs for the feasibility predictor by sampling two point clouds $(P^{obs}, P^{goal})$ from the same trajectory in the demonstration set. To find $\hat{U}^o, \hat{U}^g$, we first cluster the observation and goal point clouds into two sets $\{P_i^o\}, \{P_j^g\}$ respectively. Then, we match point clouds in the observation set to those in the goal set by finding the pairs of point clouds that are within a Chamfer distance of $\epsilon$: $\{(P_i^o, P_j^g) \mid D_{Chamfer}(P_i^o, P_j^g) < \epsilon\}$. We then remove these point clouds from the corresponding set, since these are the point clouds that have already been moved to the target location in the goal. We can then encode the remaining point clouds into $\hat{U}^o, \hat{U}^g$ as explained above. We generate hard negative samples by replacing one entity in the positive examples with a random latent vector.

**Set Cost Predictor** As we do planning in a latent space, we train a set cost predictor as our planning objective which determines how close a plan is to a given goal. The set cost predictor $C$ takes two latent set representation as input $U^o, U^g$. Since our tasks focus on matching each entity in the observation with one in the goal, we assume they have the same number of components, i.e. $|U^o| = |U^g| = N_c$. To compute the cost, we try to find the matching entity with the minimal matching cost: $C\Big(\{u_i^o\}, \{u_j^g\}\Big) = \arg\min_\sigma \sum_{i=1}^{N_c} c_\theta(u_i^o, u_{\sigma(i)}^g)$, where $\sigma$ is a permutation and $c_\theta$ is a cost prediction network parameterized by an MLP trained to predict the Chamfer Distance between the point clouds corresponding to the two latent vectors. This allows faster planning compared to first decoding latent vectors to point clouds and then computing their distance. Finally, optimization of the cost is done by performing Hungarian matching between the two sets containing latent vectors.

### 3.4 Planning with Set Representation

Given an observation and a goal point cloud $P^{obs}, P^{goal}$, we plan for the types of skills to apply in sequence, the attention for each skill (i.e. find $\hat{U}^o \subseteq U^o$), and the latent subgoals for each skill (i.e. the exact value for each latent vector in $\hat{U}^o$). As the simplest approach, we run a three-level nested optimization: In the top level, we exhaustively search over the combinations of skills to apply at each step, i.e. $k_1 \ldots, k_H$, where $k_h$ indexes the skill applied at the high-level step $h$. We only keep the sequences that end with the same set cardinality as the goal by ensuring that $\sum_{h=1}^H M_{k_h} - N_{k_h} = N_g - N_o$, where $M_{k_h}$ and $N_{k_h}$ are the number of observation and goal components for the skill applied at step $h$ and $N_o$ and $N_g$ are the number of components in the observed and target point clouds.

In the second-level optimization, we search over different attention structures. Denote the latent set at the high level step $h$ to be $U^h$. We formally define the attention structure at step $h$ to be $I^h$, which consists of a list of indices, each of length $N_{k_h}$, such that $I^h$ selects a subset from $U^{h-1}$ to be the input to the feasibility predictor, i.e. $\hat{U}^{h-1} = U_{I^h}^{h-1} \subseteq U^{h-1}$. Assume that we have $N_h$ components before applying skill $k_h$, i.e. $|U^{h-1}| = N_h$ and skill $k_h$ takes $K_h$ components as its observation. We can search over all $C_{N_h}^{K_h}$ combinations of attention structures. For components not considered by the skill, its latent vector will remain the same at step $h$. The combination of each skill attention yields an attention structure $\mathbf{I}$ for the whole plan, as illustrated in Fig. 2(c). For this level of optimization, we use a sampling-based procedure to avoid an exhaustive search over topologically equivalent attention structures. See the Appendix for how we do this efficiently.

In the low-level optimization, for each attention structure $\mathbf{I}$, we follow the optimization in DiffSkill [8]. We first sample multiple initializations for the set of latent subgoals $\mathbf{U}$, where each latent vector in the set is initialized from our generative model. We can then perform gradient descent to further optimize the latent subgoals on the following objective:

$$\arg\min_{\mathbf{k},\mathbf{I},\mathbf{U}} J(\mathbf{k},\mathbf{I},\mathbf{U}) = \prod_{h=1}^{H} f_{k_h}(\hat{U}^{h-1},\hat{U}^{h})\exp(C(U^{H},U^{g})), \tag{1}$$

where $\mathbf{k}$ is the skill sequence, $\mathbf{I}$ is the attention structure of the plan, $\mathbf{U}$ is the set of all latent subgoals, $\hat{U}^h = \{u_i^h\}_{i=1\ldots M_{k_h}}$ are the latent subgoals at step $h$, $\hat{U}^0 = \hat{U}^o \subseteq U^o$ is the attended observed set, and $U^g$ is the goal set. Finally, we can use our policy to execute our plan by following each subgoal. A summary of our method can be found in Algorithm 1.

```
Algorithm 1: Planning with Spatial and Temporal Abstraction (PASTA)
Input :Demonstration Dataset $D_{demo}$, skill horizon $T$, planning horizon $H$, modules for neural
       skill abstraction $\pi_k, f_k, r_\theta$, Point Cloud VAE with encoder $\phi$, decoder $\psi$, prior $p_u$
for each valid skill sequence $k_1, \ldots k_H$ do
    for each valid attention structure $I^1, \ldots I^H$ do
        Initialize different latent subgoals $\hat{U}^1, \ldots \hat{U}^H$ from $p_u$ ;
        Optimize latent subgoals $\hat{U}^1, \ldots \hat{U}^H$ according to Sec.  to obtain cost $J(\mathbf{k},\mathbf{I},\mathbf{U})$ ;
Choose skill sequence $\mathbf{k}$, attention structure $\mathbf{I}$, and subgoals $\mathbf{U}$ that minimizes $J(\mathbf{k},\mathbf{I},\mathbf{U})$ ;
for $h \leftarrow 0$ to $H-1$ do
    Decode the subgoal from $\hat{U}^h$ using the decoder $\psi$ ;
    Execute policy $\pi_{k_i}$ following the subgoal ;
```

## 4 Experiments

Our experiments are categorized into three parts: In Sec. 8, we describe the experimental setups and the baselines we consider. In Sec. 8 and Sec. 8, we show that PASTA outperforms the baselines and ablate different components in our framework. In Sec. 8, we demonstrate that PASTA can be effectively transferred to the real world without any fine-tuning.

### 4.1 Simulation Tasks and Baselines

**Environment setups** We consider several long-horizon dough manipulation tasks and divide them into two categories. First, we consider the three tasks from Diffskill [8]: LiftSpread, GatherTransport, and CutRearrange. These tasks require the agent to sequentially compose at most two skills to spread, cut or transport the dough. We further propose two new generalization tasks: CutRearrangeSpread (CRS) and CRS-Twice, where there are more entities during testing than during training. Similar to prior work [8], we specify the minimal planning horizon required for each task. Our approach also succeeds when we increase the horizon up to twice as long. See the Appendix, Sec 4.5 for details.

**Generalization tasks** The CRS task provides a number of demonstration trajectories performing one of the three skills: Cutting with a knife, pushing with a pusher, and spreading with a roller. The demonstration of each skill only shows a tool manipulating a single piece of dough. During testing, the agent needs to cut a dough into two, transport one piece to a spreading area and then spread it. Generalization to more entities is required as there will be two entities in the scene, whereas during training there was only one entity. Can we do even more generalization? In the CRS-Twice task, we use the same agent trained on the CRS dataset and ask it to cut two pieces of dough from a chunk, transport both of them to a spreading area, and spread them both. This is a 6-horizon task with up to 3 entities in the scene, much more complex than the skill demonstrations the agent is trained on. Due to the long-horizon nature of CRS-Twice, we specify the skill skeleton and use receding horizon planning for all the planning-based methods. See the Appendix for details.

**Baselines** We consider several baselines in simulation: First, a gradient-based trajectory optimizer with oracle information (Traj-Opt), which can solve single-stage tasks for deformable object manipulation as shown in prior works [47]. Second, a model-free RL with Soft Actor Critic [48] with RGB-D image input (SAC-Image). Third, a SAC agent that takes in the dough, target dough, and tool point clouds as input, the same as our method (SAC-Point). Fourth is DiffSkill, a model-based planning method from Lin *et al.* [8], which takes RGB-D images as input and has no spatial abstraction. The last one is Flat 3D, which extends DiffSkill to use 3D point clouds as input, with a "flat" 3D representation that encodes the whole scene to a single latent vector without any spatial abstraction.

**Metric** We specify goals as 3D point clouds of different geometric shapes such as boxes and spheres at specific locations. We report the normalized decrease in the Earth Mover Distance (EMD) approximated by the Sinkhorn diverge [49] computed as $s(t) = \frac{s_0 - s_t}{s_0}$, where $s_0, s_t$ are the initial and current EMD. We additionally set a threshold for the score to determine the success of a trial.

### 4.2 Comparison with Baselines in Simulation

| Task (Horizon) | DiffSkill tasks | | | Generalization tasks | |
|---|---|---|---|---|---|
| Method | LiftSpread (2) | GatherMove (2) | CutRearrange (3) | CRS (3) | CRS-Twice (6) |
| Traj-Opt (Oracle) [47] | 0.818 / 40% | 0.403 / 0% | 0.511 / 20% | 0.312 / 0% | 0.227 / 0% |
| SAC-Image [48] | 0.797 / 0% | 0.567 / 20% | 0.103 / 0% | 0.562 / 0% | 0.365 / 0% |
| SAC-Point [48] | 0.796 / 0% | 0.603 / 40% | 0.147 / 0% | 0.573 / 0% | 0.353 / 0% |
| DiffSkill-Image [8] | **0.920 / 100%** | 0.683 / 60% | 0.249 / 20% | -0.505 / 0% | - |
| Flat 3D (Ours) | * | * | 0.797 / 60% | -0.712 / 0% | -0.108 / 0% |
| PASTA (Ours) | 0.904 / **100%** | **0.715 / 100%** | **0.837 / 80%** | **0.896 / 100%** | **0.604 / 40%** |

**Table 1:** Normalized improvement and success rate of all methods on two sets of tasks: tasks in DiffSkill and tasks that require generalization to more steps and entities. For CRS and CRS-Twice, training data only contains skills operating on **one** component of dough but at test time there are more than two components. Only the best-performing baselines in CRS are evaluated on CRS-Twice. For LiftSpread and GatherMove, we consider the whole scene as a single spatial abstraction, so Flat 3D is equivalent to PASTA.

Table 1 shows the quantitative results of simulation tasks. First, we show that using a 3D representation is beneficial to planning and complex manipulation, as PASTA matches DiffSkill in LiftSpread and outperforms it in all the other tasks. Second, we highlight PASTA's compositional generalization power in CRS and CRS-Twice, in which there are additional components of dough at test time. Effectively using spatial abstraction to model the scene, PASTA achieves 100% success rate in CRS and retains a good performance in CRS-Twice. All the baselines, especially the planning baselines (DiffSkill, Flat 3D) fail dramatically, as they can only produce plans that consist of scenes seen in training. Impressively, PASTA is the only approach that reaches a non-zero success rate on these tasks that require compositional generalization. PASTA has a computational complexity of $\mathcal{O}(K^H \cdot |\mathbf{I}| \cdot \mathbf{T_G})$, where $K$ is the number of skills, $H$ is the planning horizon, $|\mathbf{I}|$ is the number of attention structures and $T_G$ is the time it takes to solve each gradient-based optimization. We discuss more efficient planning algorithms in the limitation section as well as in the Appendix, Sec. 5.

### 4.3 Ablation analysis

| Ablation Method | Performance / Success |
|---|---|
| No Hard Negatives Feasibility | 0.740 / 40% |
| No Sampling Planning | -0.455 / 0% |
| No Gradient Planning | 0.329 / 0% |
| PASTA (Ours) | **0.837 / 80%** |

**Table 2:** Ablation results from CutRearrange.

Table 2 shows the quantitative performance of each ablation and PASTA in CutRearrange. First, we consider a variant of feasibility predictor's training, which removes the hard negative samples and only uses random negative sampling (No Hard Negatives), which halves the success rate. Second, we consider two variants of the planner, one without gradient-descent (No Gradient Planning) and one without sampling (No Sampling Planning). The results show that both

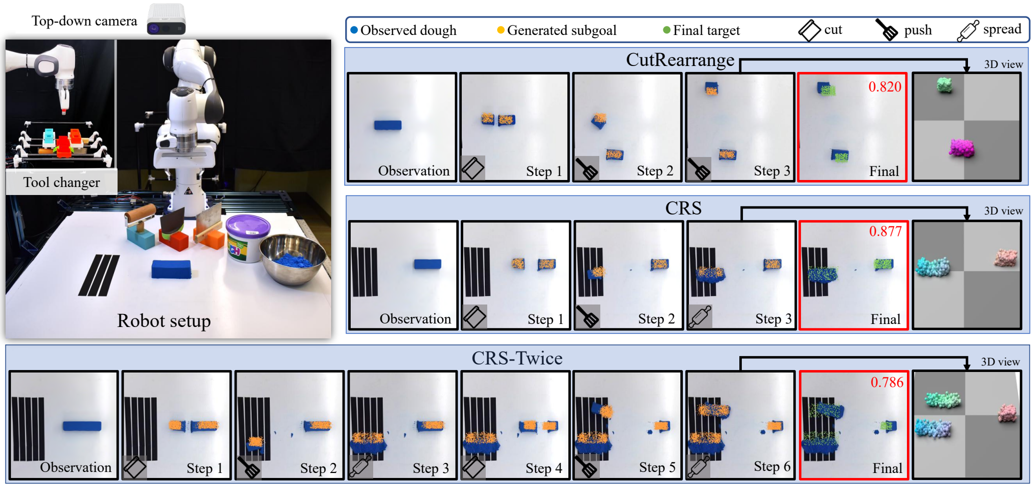


**Figure 3: Real world setup and execution with planned subgoals**. Our workspace consists of a Franka robot, a top-down camera, and a novel tool changer behind the robot that allows the robot to automatically switch tools. For each task, we show frames after executing a skill overlaid with the decoded point cloud subgoal; we report the final performance in red and overlay the ground truth target in green in the final frame. Additionally, we include a 3D view of the last generated subgoal to show the shape variations.

components are crucial to planning. We provide more ablations on the policy and the planner in the Appendix.

### 4.4 Real World Experiments

Figure 3 shows our real world setup. We use a Franka robot with an Azure Kinect camera capturing the RGB-D observation of the workspace. The robot is equipped with a tool station [50] that allows an automatic change of tools. For real world "dough", we use Kinect Sand as a proxy because of its stable physical property. We transfer the feasibility predictor and cost predictor of PASTA directly from simulation and define heuristic controllers for the skills. For evaluation, we first generate a desired target point cloud and then reset the dough to its initial shape and record its point cloud. We then plan using PASTA and then use the heuristic controllers to follow the plan. Finally, we report the normalized improvement EMD. We compare with the Flat3D method and also with human.

We evaluate three of the simulation tasks: CutRearrange, CRS, and CRS-Twice. For each task, we evaluate the same four initial and target shapes for all methods and report the performance in Table 3. Figure 3 shows the keyframes from the execution of PASTA. We overlay the planned subgoals as well as the final goal. PASTA performs on par with human in the real world, highlighting the robustness of our planner and the advantage of using 3D representation for sim2real transfer.

| Task (Horizon)<br>Method | CutRearrange (3) | CRS (3) | CRS-Twice (6) |
|---|---|---|---|
| Flat 3D | 0.351 ± 0.478 | 0.007 ± 0.429 | - |
| PASTA (Ours) | **0.836 ± 0.029** | **0.854 ± 0.016** | **0.795 ± 0.035** |
| Human | 0.910 ± 0.014 | 0.863 ± 0.018 | 0.895 ± 0.013 |

**Table 3:** Normalized improvement on real world tasks. Each entry shows the mean and std of the performance over 4 runs. Flat 3D does not produce any meaningful plan for CRS, so we do not evaluate it on CRS-Twice.

## 5 Conclusions and Limitations

In this work, we propose a planning framework named PASTA that incorporates both spatial and temporal abstraction by planning with a 3D latent set representation with attention structure. We demonstrate a manipulation system in the real world that uses PASTA to plan with multiple tool-use skills to solve the challenging deformable object manipulation tasks, and we show that it significantly outperforms a flat 3D representation, especially when generalizing to more complex tasks.

**Limitations:** First, we only transfer the planner to the real world and use heuristic controllers instead of the policy trained in simulation. This is due to the sim2real gap caused by the differences in dough's physical parameters, table friction, and occlusions from the robot arm. Prior work [38] shows promising results in transferring a closed-loop policy taking partial point clouds as input, and future work can explore better sim2real methods. Second, our planner exhaustively searches overall skill combinations and attention structures and does not scale well to longer sequences with more skills. Future work can incorporate more efficient search algorithms or priors to prune the search space. Finally, we rely on an unsupervised clustering method for entity decomposition and a point cloud VAE for mapping an observation to our latent set representation. Future work can incorporate self-supervised methods for learning the decomposition. For further discussion on the limitations and future work, please see Sec. 5 of the Appendix.

### Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. IIS-2046491, IIS-1849154, NSF award under AWD00001520, LG Electronics, Carnegie Mellon University's GSA/Provost GuSH Grant funding, and Amazon Research Award.

## References

[1] X. Ma, D. Hsu, and W. S. Lee. Learning latent graph dynamics for visual manipulation of deformable objects. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8266–8273. IEEE, 2022.

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

[3] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019.

[4] X. Lin, Y. Wang, Z. Huang, and D. Held. Learning visible connectivity dynamics for cloth smoothing. In *Conference on Robot Learning*, 2021.

[5] P. R. Florence, L. Manuelli, and R. Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. In *Conference on Robot Learning*, pages 373–385. PMLR, 2018.

[6] Y. Li, S. Li, V. Sitzmann, P. Agrawal, and A. Torralba. 3d neural scene representations for visuomotor control. In *Conference on Robot Learning*, pages 112–123. PMLR, 2022.

[7] D. Driess, Z. Huang, Y. Li, R. Tedrake, and M. Toussaint. Learning multi-object dynamics with compositional neural radiance fields. In *International Symposium of Robotics Research*, 2022.

[8] X. Lin, Z. Huang, Y. Li, J. B. Tenenbaum, D. Held, and C. Gan. Diffskill: Skill abstraction from differentiable physics for deformable object manipulations with tools. *ICLR*, 2022.

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

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

[11] 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.

[12] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. *Robotics: Science and Systems Foundation*, 2018.

[13] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[14] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl-the planning domain definition language. *Technical Report CVC TR-98-003/DCS TR-1165*, 1998.

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

[16] W. Yuan, C. Paxton, K. Desingh, and D. Fox. Sornet: Spatial object-centric representations for sequential manipulation. In *Conference on Robot Learning*, pages 148–157. PMLR, 2022.

[17] J. Liang, M. Sharma, A. LaGrassa, S. Vats, S. Saxena, and O. Kroemer. Search-based task planning with learned skill effect models for lifelong robotic manipulation. In *2022 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022.

[18] E. Ugur and J. Piater. Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2627–2633, 2015. doi:10.1109/ICRA.2015.7139553.

[19] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez. Learning compositional models of robot skills for task and motion planning. *The International Journal of Robotics Research*, 40(6-7):866–894, 2021.

[20] 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.

[21] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[22] K. Fang, Y. Zhu, A. Garg, S. Savarese, and L. Fei-Fei. Dynamics learning with cascaded variational inference for multi-step manipulation. In *Conference on Robot Learning*, pages 42–52. PMLR, 2020.

[23] A. Simeonov, Y. Du, B. Kim, F. Hogan, J. Tenenbaum, P. Agrawal, and A. Rodriguez. A long horizon planning framework for manipulating rigid pointcloud objects. In *4th Conference on Robot Learning (CoRL 2020)*. Massachusetts Institute of Technology, 2020.

[24] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2020.

[25] X. Ma, D. Hsu, and W. S. Lee. Learning latent graph dynamics for deformable object manipulation. *arXiv preprint arXiv:2104.12149*, 2021.

[26] N. Heravi, A. Wahid, C. Lynch, P. Florence, T. Armstrong, J. Tompson, P. Sermanet, J. Bohg, and D. Dwibedi. Visuomotor control in multi-object scenes using object-aware representations. *arXiv preprint arXiv:2205.06333*, 2022.

[27] C. Devin, P. Abbeel, T. Darrell, and S. Levine. Deep object-centric representations for generalizable robot learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7111–7118. IEEE, 2018.

[28] D. Xu, R. Martín-Martín, D.-A. Huang, Y. Zhu, S. Savarese, and L. F. Fei-Fei. Regression planning networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[29] C. Schenck and D. Fox. Visual closed-loop control for pouring liquids. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2629–2636. IEEE, 2017.

[30] G. Narasimhan, K. Zhang, B. Eisner, X. Lin, and D. Held. Transparent liquid segmentation for robotic pouring. In *2022 International Conference on Robotics and Automation (ICRA)*, 2022.

[31] P. Sundaresan, J. Grannen, B. Thananjeyan, A. Balakrishna, M. Laskey, K. Stone, J. E. Gonzalez, and K. Goldberg. Learning rope manipulation policies using dense object descriptors trained on synthetic depth data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9411–9418. IEEE, 2020.

[32] P. Mitrano, D. McConachie, and D. Berenson. Learning where to trust unreliable models in an unstructured world for deformable object manipulation. *Science Robotics*, 6(54):eabd8170, 2021.

[33] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *2010 IEEE International Conference on Robotics and Automation*, pages 2308–2315. IEEE, 2010. URL https://ieeexplore.ieee.org/abstract/document/5509439.

[34] R. Hoque, D. Seita, A. Balakrishna, A. Ganapathi, A. Tanwani, N. Jamali, K. Yamane, S. Iba, and K. Goldberg. VisuoSpatial Foresight for Physical Sequential Fabric Manipulation. *Autonomous Robots (AURO) journal*, 2021.

[35] Z. Huang, X. Lin, and D. Held. Mesh-based dynamics model with occlusion reasoning for cloth manipulation. *Robotics: Science and Systems (RSS)*, 2022.

[36] T. Weng, S. Bajracharya, Y. Wang, K. Agrawal, and D. Held. Fabricflownet: Bimanual cloth manipulation with a flow-based policy. In *Conference on Robot Learning*, 2021.

[37] H. Shi, H. Xu, Z. Huang, Y. Li, and J. Wu. Robocraft: Learning to see, simulate, and shape elasto-plastic objects with graph networks. *Robotics: Science and Systems (RSS)*, 2022.

[38] C. Qi, X. Lin, and D. Held. Learning closed-loop dough manipulation using a differentiable reset module. *IEEE Robotics and Automation Letters*, 7(4):9857–9864, 2022. doi:10.1109/LRA.2022.3191239.

[39] N. Figueroa, A. L. P. Ureche, and A. Billard. Learning complex sequential tasks from demonstration: A pizza dough rolling case study. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 611–612. Ieee, 2016.

[40] C. Matl and R. Bajcsy. Deformable elasto-plastic object shaping using an elastic hand and model-based reinforcement learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3955–3962. IEEE, 2021.

[41] E. Heiden, M. Macklin, Y. S. Narang, D. Fox, A. Garg, and F. Ramos. DiSECt: A Differentiable Simulation Engine for Autonomous Robotic Cutting. In *Proceedings of Robotics: Science and Systems*, July 2021. doi:10.15607/RSS.2021.XVII.067.

[42] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[43] F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf. Object-centric learning with slot attention. *Advances in Neural Information Processing Systems*, 33:11525–11538, 2020.

[44] G. Yang, X. Huang, Z. Hao, M.-Y. Liu, S. Belongie, and B. Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. *ICCV*, 2019.

[45] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[46] Y. Lee, E. S. Hu, and J. J. Lim. Ikea furniture assembly environment for long-horizon complex manipulation tasks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6343–6349. IEEE, 2021.

[47] Z. Huang, Y. Hu, T. Du, S. Zhou, H. Su, J. B. Tenenbaum, and C. Gan. Plasticinelab: A soft-body manipulation benchmark with differentiable physics. In *International Conference on Learning Representations*, 2020.

[48] 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.

[49] T. Séjourné, J. Feydy, F.-X. Vialard, A. Trouvé, and G. Peyré. Sinkhorn divergences for unbalanced optimal transport. *arXiv preprint arXiv:1910.12958*, 2019.

[50] Y. Zhang. Library of behaviors and tools for robot manipulation. Master's thesis, Carnegie Mellon University, Pittsburgh, PA, August 2022.

[51] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

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

[53] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[54] C. Garrett, T. Lozano-Pérez, and L. Kaelbling. Sample-based methods for factored task and motion planning. 2017.

[55] B. Kim and L. Shimanuki. Learning value functions with relational state representations for guiding task-and-motion planning. In *Conference on Robot Learning*, pages 955–968. PMLR, 2020.

[56] 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.

[57] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[58] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022.

[59] W. Yu, J. Tan, C. K. Liu, and G. Turk. Preparing for the unknown: Learning a universal policy with online system identification. *arXiv preprint arXiv:1702.02453*, 2017.

[60] A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[61] F. Ramos, R. C. Possas, and D. Fox. Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. *arXiv preprint arXiv:1906.01728*, 2019.

[62] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

[63] C. Qi, X. Lin, and D. Held. Learning closed-loop dough manipulation using a differentiable reset module. *IEEE Robotics and Automation Letters*, 7(4):9857–9864, 2022.

# Appendix

## Contents

# 1 Implementation Details

## 1.1 Entity Encoding

To train our point cloud variational autoencoder [44], we normalize the point cloud of each entity $P_i$ to be centered at the origin, i.e. $\bar{P}_i = P_i - t_i$, where $t_i$ is the mean of all points in $P_i$. We then encode each centered $\bar{P}_i$ into a latent encoding: $z_i = \phi(\bar{P}_i)$. Our latent representation $u_i = [z_i, t_i]$ consists of the encoding of the point cloud's shape $z_i$ and the position of the center of the point cloud $t_i$. We model the point cloud's position $t_i$ explicitly such that the learned latent embedding $z_i$ can focus on shape variation alone and the model that plans over $u_i$ can still reason over point clouds at different spatial locations. During training, we record the 3D bounding box of all training data $t_{min}, t_{max} \in \mathbb{R}^3$, and we sample from $[t_{min}, t_{max}]$ during planning. We denote this combined distribution of $u = [z, t]$ as $P_u$.

## 1.2 Details on Training Set Feasibility Predictor

**Hard Negative Samples** Suppose skill $k$ takes $N_k$ latent vectors from observation $\hat{U}^o$ and $M_k$ latent vectors from goal $\hat{U}^g$ as input. To generate random pairs of observations and goals as negative samples for the feasibility predictor, we can sample each latent point cloud representation $u_i$ by sampling the shape $z_i$ from the VAE prior $p_z$ and sampling the position $t_i$ from the distribution of positions in the training dataset. Such random negative samples are used similarly in DiffSkill [8]. However, as the combined dimension of the set representation becomes larger compared to a flat representation, we need a way to generate harder negative samples. To do so, for a positive pair of set representation $(\{u_i^o\}, \{u_j^g\})$, we randomly replace one of the entities $u_i^o$ or $u_j^g$ with a random sample in the latent space and use it as a negative sample. Our ablation results show that this way of generating hard negative samples is crucial for training our set feasibility predictor.

**Noise on Latent Vectors** During training of the feasibility predictor, for each of the input latent vector $u = [z, t]$, where $z \in \mathbb{R}^{D_z}$ is the latent encoding of the shape and $t \in R^3$ is the 3D position of the point cloud, we add a Gaussian noise to each part, i.e. $\hat{z} = z + \sigma_z \epsilon$ and $\hat{t} = t + \sigma_t \epsilon$, where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The amount of noise determines the smoothness of the feasibility landscape. Without any noise, planning with gradient descent with the feasibility function becomes much harder.

## 1.3 Details on the Attention Structure for Planning with Set Representation

Given the initial latent set observation $U^{obs}$ with $N_o$ components, and the skill sequences $k_1, \ldots, k_H$, in this section we describe how to generate the attention structure. We denote the latent set representation at step $h$ as $U^h$, $h = 1 \ldots H$ and define $U^0 = U^{obs}$. As skill $k_h$ takes in $N_{k_h}$ components as input and $M_{k_h}$ components as output, by calculation we know that $U^h$ has $N_o + \sum_{i=1}^h M_{k_i} - N_{k_i}$ components. From now on, we denote $|U^h| = N_h$ and $U^h = \{u_{h,1}, \ldots, u_{h,N_h}\}$. As the skill $k_h$ only applies to a subset of the input $U^{h-1}$, we now formally define the attention structure at step $h$ to be $I^h$, which consists of a list of indices, each of length $N_{k_h}$, such that $I^h$ selects a subset from $U^{h-1}$ to be the input to the feasibility predictor, i.e.

$$\hat{U}^{h-1} = U_{I^h}^{h-1} \subseteq U^{h-1}.$$

However, enumerating all $I^h$ is infeasible, as there are $C_{N_h}^{N_{k_h}}$ combinations for each step. Fortunately, we do not have to enumerate all different structures. The insight here is that, for each attention structure $I_1, \ldots I_H$, we will perform a low-level optimization. In this low-level optimization, we will first initialize all the latent vectors to be optimized from $P_u$ and then perform gradient descent on them. As many of the attention structures yield topologically equivalent tree structures (An example of such a tree is illustrated in Figure 2c of the main paper), and each latent vector in the tree is sampled independently from the same distribution $P_u$, these topologically equivalent tree structures result in the same optimization process. As such, we do not need to exhaust all of such attention structures.

Instead of enumerating each topologically different structure and then sampling multiple initializations for the low-level optimization, we randomly sample sequences $(I_1, \dots I_H)$ and perform low-level gradient-descent optimization on all the samples. In this way, with enough samples, we will be able to cover all attention structures.

Now, we can sequentially build up the subgoals latent set representation during planning. Specifically, assuming that we have constructed the previous latent set representation $U^{h-1}$, we will now describe the procedure for constructing $U^h$, as well as the predicted feasibility for the current skill $k_h$, i.e. $f_{k_h}(\hat{U}^{o,h}, \hat{U}^{g,h})$, where $\hat{U}^{o,h}, \hat{U}^{g,h}$ are the subset of $U^{h-1}$ and $U^h$ attended by the feasibility predictor. First, we generate $I_h$ by randomly choosing the index of a subset from $U_{h-1}$. $U^h$ are composed of two parts: The first part is the latent vectors generated by applying the skill $k_h$. For this part, we will create a set of new vectors $u_{h,0}, \dots u_{h,M_{h_k}}$. This part of the latent vectors will be attended by the feasibility predictor as $\hat{U}^{g,h} = \{u_{h,0}, \dots u_{h,M_{h_k}}\}$. The second part of $U^h$ comes from the previous latent set vectors that are not modified by the skill, i.e. $U^{h-1} \setminus U^{h-1}_{I_h}$, and $U^h$ is the addition of both parts, i.e.

$$U^h = \hat{U}^{g,h} \cup (U^{h-1} \setminus U^{h-1}_{I_h})$$

In this way, we can sequentially build up $U^h$ from $U^{h-1}$, and $U^0$ is simply $U^{obs}$. At the same time, we have determined our attention structure and the feasibility prediction. Our objective can thus be written as

$$\arg\min_{\mathbf{k},\mathbf{I},\mathbf{U}} J(\mathbf{k}, \mathbf{I}, \mathbf{U}) = \prod_{h=1}^{H} f_{k_h}(\hat{U}^{o,h}, \hat{U}^{g,h}) \exp(-C(U^H, U^g)), \tag{2}$$

where $\mathbf{U}$ is the set union of all latent vectors to be optimized.

### 1.4 Network Architectures

**Set Feasibility Predictor** We use a Multi-Layer Perceptron (MLP) with ReLU activations for our feasibility predictor. We apply max-pooling to the transformed latent vectors of $\hat{U}^o$ and $\hat{U}^g$ to achieve permutation invariance. Below is our architecture:

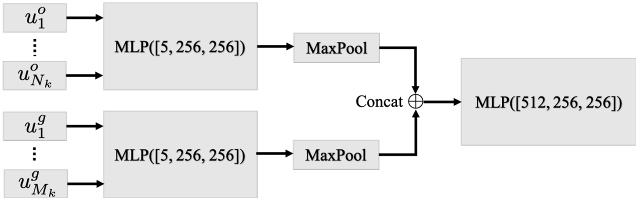


**Figure 4:** Architecture for the set feasibility predictor

**Set Cost Predictor** We use a 3-layer MLP with a hidden dimension of 1024 and ReLU activations.

**Set Policy** The set point cloud policy for the $k^{th}$ skill $\pi_k$ takes in an observed point cloud $P^{obs}$, a goal point cloud $P^{goal}$, and a tool point cloud $P^{tool}_k$ and outputs an action at each timestep to control the tool directly. The tool point cloud $P^{tool}$ is obtained by sampling points on the mesh surface of the tool and then transforming these points to the same camera frame as the $P^{obs}$ and $P^{goal}$, assuming the pose of the tool is known from the robot state. Instead of taking latent vectors as input, the policy functions directly in point cloud space, which allows it to handle times when spatial abstraction is ambiguous. For instance, during cutting and merging, the number of dough components gradually increases or decrease, during which the latent set representation is not changing smoothly while the point clouds change smoothly during the process. We later show the advantage of using point clouds directly as the policy input. We concatenate each point's $(x, y, z)$ coordinates with a one-hot encoding to indicate whether the point belongs to the observation, tool, or goal, and we input the points into a PointNet++ [51] encoder followed by an MLP which outputs the action. We use a point

|  | LiftSpread | GatherMove | CutRearrange | CRS + CRS-Twice |
|---|---|---|---|---|
| # of initial configurations | 200 | 200 | 1500 | 1200 |
| # of target configurations | 200 | 200 | 1500 | 1200 |
| # of training trajectories | 1800 | 1800 | 1350 | 1080 |
| # of testing trajectories | 200 | 200 | 150 | 120 |
| # of total trajectories | 2000 | 2000 | 1500 | 1200 |
| # of total transitions | 1e5 | 1e5 | 7.5e4 | 6e4 |

**Table 4:** Summary of training/testing data

cloud for the tool to allow the PointNet encoder to reason about the interaction between the tool and the dough in the same space. We use PyTorch Geometric's [52] implementation of PointNet++ and with the following list of modules in our encoder.

```
SAModule(0.5,0.05,MLP([3+3,64,64,128]))
SAModule(0.25,0.1,MLP([128+3,128,128,256]))
GlobalSAModule(MLP([256+3,256,128,512,1024]))
```

The MLP following the encoder consists of hidden dimensions $[1024, 512, 256]$ and ReLU activations.

### 1.5 Training Details

**Training data.** We inherit the data generation procedure from DiffSkill [8]: first, we randomly generate initial and target configurations. The variations in these configurations include the location, shape, and size of the dough and the location of the tool. We then sample a specific initial configuration and a target configuration and perform gradient-based trajectory optimization to obtain demonstration data. For each task, the demonstration data consists of all the transitions from executing the actions outputted by the trajectory optimizer. We perform a train/test split on the dataset and select 5 configurations in the test split for evaluating the performance for all the methods. More information about training and testing data can be found in Table 4.

**Point cloud VAE.** We train our point cloud VAE by maximizing the evidence lower bound (ELBO). For a dataset of observations $P(X)$, which consists of the segmented point cloud of each entity in the scene, we optimize the following objective:

$$\mathcal{L}_{VAE} = \mathbb{E}_{Q_\phi(z|x)}\left[\log P_\psi(X|z)\right] - D_{KL}(Q_\phi(z|X)||p(z)) \tag{3}$$

where $Q_\phi(z|X)$ is the encoder modeled as a diagonal Gaussian, $P_\psi(X|z)$ is the decoder, and $p(z)$ is standard Gaussian. The VAE is pretrained, and we fix its weights prior to training the other modules.

**Point cloud policy.** We train our point cloud policy with standard behavioral cloning (BC) loss, i.e. for the $k$-th skill, we sample a transition from the demonstration data, which contains the observed point clouds $\{P_i^o\}$, goal point clouds $\{P_j^g\}$, a tool point cloud $P_k^{tool}$, and the action of the tool $a$. Then, we match point clouds in the observation set to those in the goal set by finding the pairs of point clouds that are within a Chamfer Distance of $\epsilon$: $\{(P_i^o, P_j^g) \mid D_{Chamfer}(P_i^o, P_j^g) < \epsilon\}$ and filter out the non-relevant point clouds. Last, we pass the filtered point clouds into the policy and minimize the following loss:

$$\mathcal{L}_{\pi_k} = \mathbb{E}\left[\|a - \pi_k(\{P_i^o\}, P_k^{tool}, \{P_j^g\})\|^2\right] \tag{4}$$

**Feasibility predictor.** We train the feasibility predictor for the $k$-th skill by regressing to the ground-truth feasibility label using mean squared error (MSE) as loss, i.e.

$$\mathcal{L}_{f_k} = \mathbb{E}\left[\left(f_k(\hat{U}^o, \hat{U}^g) - \mathbb{1}\{\hat{U}^o, \hat{U}^g \text{ is a positive pair}\}\right)^2\right] \tag{5}$$

During training, we obtain positive pairs for the feasibility predictor by sampling two point clouds $(P^{obs}, P^{goal})$ from the same trajectory in the demonstration set. To find $\hat{U}^o, \hat{U}^g$, we first cluster the

| Parameter | LiftSpread | GatherMove | CutRearrange | CRS + CRS-Twice |
|---|---|---|---|---|
| Yield stress | 200 | 200 | 150 | 150 |
| Ground friction | 1.5 | 1.5 | 0.5 | 0.5 |
| Young's modulus (E) | 5e3 | 5e3 | 5e3 | 5e3 |
| Poisson's ratio ($\nu$) | 0.15 | 0.15 | 0.15 | 0.15 |

**Table 5:** Parameters for simulation dough

observation and goal point clouds into two sets $\{P_i^o\}, \{P_j^g\}$ respectively. Then, we match point clouds in the observation set to those in the goal set by finding the pairs of point clouds that are within a Chamfer Distance of $\epsilon$: $\{(P_i^o, P_j^g) \mid D_{Chamfer}(P_i^o, P_j^g) < \epsilon\}$. We then remove these point clouds from the corresponding set, since these are the point clouds that have already been moved to the target location in the goal. We can then encode the remaining point clouds into $\hat{U}^o, \hat{U}^g$ using our VAE.

**Cost predictor.** We train the cost predictor by simply regressing to the Chamfer Distance (CD) between two entities represented by their latent vectors, i.e.

$$\mathcal{L}_c = \mathbb{E}\left[\left(c(\phi(P_i), \phi(P_j)) - D_{Chamfer}(P_i, P_j)\right)^2\right] \tag{6}$$

where $P_i$ and $P_j$ are point clouds of a single entity sampled from the dataset and $\phi$ is the encoder. There are two reasons that we train a cost predictor on latent vectors instead of directly computing the Chamfer Distance between two point clouds. For one, decoding each latent vector would greatly bottleneck the planning speed. Experiments on CutRearrange show that with our learned cost predictor, the planning takes 35s; on the other hand, if we decode the latent vectors and use the Chamfer Distance, even with a subsampled point cloud of 200 points, the planning takes 37200s (around 10 hours), which is impractical to use. Moreover, using a cost predictor can also offer us the flexibility to incorporate more complex reward functions in the future.

Finally, We train our policy, feasibility predictor, and cost predictor with the following loss:

$$\mathcal{L}_{PASTA} = \sum_{k=1}^{K} \mathbb{E}\left[\lambda_\pi \mathcal{L}_{\pi_k} + \lambda_f \mathcal{L}_{f_k} + \lambda_c \mathcal{L}_c\right] \tag{7}$$

We use $\lambda_\pi = 1$, $\lambda_f = 10$, and $\lambda_c = 1$ for all of our experiments.

# 2 Details on Simulation Experiments

## 2.1 Hyperparameters for Simulation Dough

We use PlasticineLab [47] for evaluating our simulation experiments. We provide the hyperparameters that are relevant to the properties of the dough in simulation to enhance the replicability of our results. See Table 5 for details.

## 2.2 Hyperparameters for DBSCAN

To cluster a point cloud, we use Scikit-learn's [53] implementation of DBSCAN [42] with `eps=0.03, min_samples=6, min_points=10` for all of our environments. Further, we assign each noise point identified by DBSCAN to its closest cluster.

## 2.3 Hyperparameters for PASTA

Table 6 shows the hyperparameters used for PASTA in our simulation tasks. Planning for CRS-Twice requires a large amount of samples. Therefore, we modify the planner to improve sample efficiency. See Sec. 2.4 for details.

| Training parameters | LiftSpread | GatherMove | CutRearrange | CRS | CRS-Twice |
|---|---|---|---|---|---|
| *Point Cloud VAE* | | | | | |
| learning rate | 2e-3 | 2e-3 | 2e-3 | 2e-3 | 2e-3 |
| latent dimension | 2 | 2 | 2 | 2 | 2 |
| *Feasibility predictor* | | | | | |
| learning rate | 1e-4 | 1e-3 | 1e-4 | 1e-4 | 1e-4 |
| batch size | 256 | 256 | 256 | 256 | 256 |
| noise on shape encoding $\sigma_z$ | 0 | 0 | 0 | 0.02 | 0.02 |
| noise on position $\sigma_t$ | 0.01 | 0.01 | 0.005 | 0.01 | 0.01 |
| *Cost predictor* | | | | | |
| learning rate | 1e-4 | 1e-3 | 1e-4 | 1e-4 | 1e-4 |
| batch size | 256 | 256 | 256 | 256 | 256 |
| *Policy* | | | | | |
| learning rate | 1e-4 | 1e-3 | 1e-4 | 1e-4 | 1e-4 |
| batch size | 10 | 10 | 10 | 10 | 10 |
| noise on point cloud | 0.005 | 0.005 | 0.005 | 0.005 | 0.005 |
| Planning parameters | LiftSpread | GatherMove | CutRearrange | CRS | CRS-Twice |
| learning rate | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| number of iterations | 200 | 100 | 100 | 200 | 300 |
| number of samples | 5000 | 5000 | 5000 | 50000 | 500000 |

**Table 6:** Summary of hyperparameters used in PASTA. For CRS-Twice, we use the same model as CRS but modify the planner to have better sample efficiency.

### 2.4 Receding Horizon Planning for CRS-Twice

As the planning horizon increases, the number of possible skill sequences as well as the number of possible attention structures increases exponentially. The task of CRS-Twice has a planning horizon of 6 and is a much more difficult task to solve. As such, for this task, we specify the skill sequences and use Receding Horizon Planning (RHP). Starting from the first time step, we follow the procedure in Algorithm 1 but only optimize for $H_{RHP}$ steps into the future and compare the achieved subgoal with the final target to compute the planning loss. After optimization, we take the first subgoal from the plan and discard the rest of the plan. We then repeat this process until we reach the overall planning horizon $H$. In our experiments, we use $H_{RHP} = 3$. While we can perform model predictive control and execute the first step before planning for the second step, we find this open-loop planning and execution to be sufficient for the task.

## 3 Details on Real World Experiments

### 3.1 Heuristic Policies

Transferring the learned policies from simulation to the real world can be more difficult than transferring the planner itself, as the policies are affected more by the sim2real gap, such as the difference in friction and properties of dough in the real world. To sidestep this challenge, for our real world experiments, we design three heuristic policies: cut, push, and roll to execute the generated plans in the real world.

Just like our learned policies in simulation, each heuristic policy takes in the current observation and the generated subgoal in point clouds and outputs a sequence of desired end effector positions used for impedance control. In addition, each policy takes in the attention mask provided by the planner indicating the components of interest. The same DBSCAN procedure is used for this. The cut policy first calculates the cutting point by computing the length ratio of the resulting components. Then it cuts the dough and separates it such that the center of mass of each resulting component matches the one in the subgoal. For the push policy, given a component and a goal component, the policy pushes the dough in the direction that connects the two components' center of mass. The roll policy first moves the roller down to make contact with the dough. Then, based on the goal component's length, the policy calculates the distance it needs to move the roller back and forth when making contact with the dough.

### 3.2 Procedure for Resetting the Dough

To compare different methods with the same initial and target configurations, we first use a 3D-printed mold to fit the dough to the same initial shape. We then overlay the desired initial location on the image captured by the top-down camera and place the dough at the corresponding location in the workspace to ensure different methods start from the same initial location.

### 3.3 Procedure for the Human Baseline

Following the same procedure in section 3.2, we first reset the dough to the initial configuration. Then, we overlay the goal point cloud on the image captured by the top-down camera. The overlay image is shown on a screen and presented to the human in real-time when the human is completing the task.

### 3.4 Procedure for Making the Real Dough

| Material | Quantity(g) | Baker's percentage(%) |
|---|---|---|
| Flour | 300 | 100 |
| Water | 180 | 60 |
| Yeast | 3 | 1 |

**Table 7:** All-purpose dough recipe

We follow the recipe shown in Table 7 to make the real dough. Following the tradition of baking, we use the backer's percentage, so that each ingredient in a formula is expressed as a percentage of the flour weight, and the flour weight is always expressed as 100%. First, we take 300 grams of flour, 3 grams of yeast, and 180 grams of water into a basin. Then, we mix the ingredients and knead the dough for a few minutes. Next, we use a food warp to seal the dough in the basin and put them in the refrigerator to let the dough rest for 4-5 hours. Finally, we take out the dough from the refrigerator and reheat it with a microwave for 30-60 seconds to soften it.

## 4 Additional Experiments

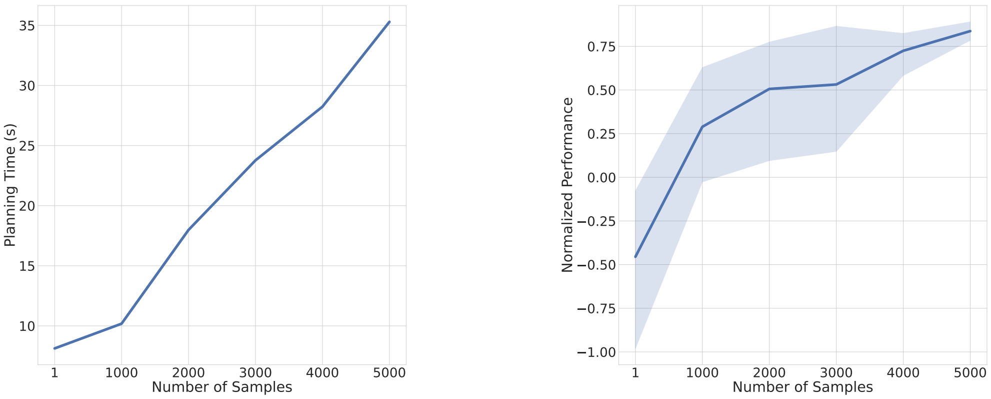
### 4.1 Ablation Studies

**Ablations on feasibility predictor.** Following the discussions in Sec 1.2, we train a feasibility predictor without adding any noise to show that adding noise helps with the optimization landscape during planning. We call this ablation *No Smoothing Feasibility*. As shown in Table 8, this variant only achieves half of the success rate of PASTA, suggesting the importance of noise during training.

| Ablation Method | Performance / Success |
|---|---|
| No Smoothing Feasibility | 0.744 / 40% |
| Shared Encoder Policy | -0.304 / 0% |
| Tool Concat Policy | 0.516 / 60% |
| Set without Filtering | 0.360 / 20% |
| PASTA (Ours) | **0.837 / 80%** |

**Table 8:** Additional ablation results from CutRearrange.

**Ablations on policy.** We consider two ablation methods for our set policy. First, we consider a *Shared Encoder Policy* that takes in the latent vectors from the encoder and uses a max pooling layer followed by an MLP to produce the action. The architecture is very similar to our Set Feasibility Predictor. Our results in Table 8 show that this architecture has zero success in our task. We hypothesize that this is because the entity encoding can



**(a)** Planning time v.s. number of samples

**(b)** Planning performance v.s. number of samples

**Figure 5:** Planning time and performance with varying number of samples in CutRearrange.We show the mean and standard deviation of performance over 5 runs.

be unstable during the skill execution. For example, during cutting, the dough slowly transitions from one piece to two pieces, making the input to the policy unstable.

Second, we compare with a *Tool Concat Policy* that takes in the observation and goal point cloud of the dough, passes them through a PointNet++ [51] encoder to produce a feature, and then concatenates the tool state to the feature. The concatenated feature is passed through a final MLP to output the action. In comparison, the set policy in PASTA takes the point cloud of the tool and concatenates it with the dough in the point cloud space before passing it to the PointNet. We hypothesize that this way allows PointNet to reason more easily about the spatial relationships between the tool point cloud and the dough point cloud. Results in Table 8 highlight the advantage of using a point cloud to represent the tool.

**Ablation on set representation.** We consider a variant of PASTA *Set without Filtering*, which uses the same set representation as PASTA, but does not filter entities that are approximately the same both during training and testing. This filtering is only possible with a set representation and we want to show the advantage of this filtering. For this ablation, during training, the feasibility predictor takes in all the entities in the scene in set representation, and the policy takes in the concatenation of point clouds from each entity. During planning, we do not enumerate attention structures but instead optimize for all the entities. As shown in Table 8, without filtering, this ablation performs significantly worse than PASTA, showing that filtering is an important advantage enabled by our set representation.

## 4.2 Visualization of the Latent Space

We visualize the latent space of PASTA in CutRearrange in Figure 6 and visualize the latent space of Flat 3D baseline in Figure 7 for comparison. Since we use a latent dimension of 2 for all of our environments, we can visualize the original latent space without applying any dimensionality reduction techniques. PASTA only encodes the shape of each entity and thus can better model the variations in shapes. On the other hand, Flat 3D couples the shape variation with the relative position of two entities. This makes a flat representation difficult to generalize compositionally to scenes with different numbers of entities or scenes with entities that have novel relative spatial locations to each other.

### 4.3 Runtime of PASTA

We implement the planning in the latent set representation in an efficient way, which can plan with multiple different structures in parallel on a GPU. To demonstrate the efficiency of PASTA, we vary the number of samples used for planning and record the planning time and final performance. We conduct the experiments in CutRearrange. Figure 5a shows that the planning time scales approximately linearly with the number of samples, and Figure 5b shows the planning performance versus the number of samples. As the result suggests, PASTA can achieve its optimal performance with a very short amount of planning time (under 1 minute) for the majority of our tasks. Finally, we summarize the planning time for all of our tasks in simulation in Table 9.

| | LiftSpread | GatherMove | CutRearrange | CRS | CRS-Twice* |
|---|---|---|---|---|---|
| Planning time (seconds) | 58 | 35 | 35 | 307 | 7810 |

**Table 9:** Summary of planning time of PASTA in all of the simulation tasks. CRS-Twice uses Receding Horizon Planning, which results in an increase in planning time.

### 4.4 Additional metrics for real world experiments

We also quantitatively computed the action error v.s. subgoal error for our real world trajectories. The results are shown in Table 10. From the results in the table, our planned goal is closer to the ground truth goal than the achieved goal, measured by the Earth Mover's Distance (EMD), which shows that the controller does not compensate for the error of the planner.

| | CutRearrange | CRS | CRS-Twice |
|---|---|---|---|
| EMD(planned goal, ground-truth goal) | $0.038 \pm 0.004$ | $0.027 \pm 0.004$ | $0.029 \pm 0.002$ |
| EMD(reached goal, ground-truth goal) | $0.056 \pm 0.007$ | $0.044 \pm 0.006$ | $0.054 \pm 0.016$ |

**Table 10:** Action error v.s. subgoal error for real world experiments. For each task, the mean ± std for 4 trajectories are shown.

### 4.5 Robustness of PASTA

We show that PASTA is robust to two types of variations and can retain high performance.

**Robust to planning horizon** First, we increase the planning horizon from the minimal length for the task (3) to twice the minimal length (6), and we observe that PASTA retains a high performance across all horizons. The results are shown in Table 11. This suggests that in practice, one can specify a maximum planning horizon for PASTA when the exact horizon is unknown.

**Robust to distractors** Second, we show that PASTA is robust to distractors in the scene. We add 2 distractor objects in CRS (which makes the scene have 4 objects in total). We observe that PASTA retains a normalized performance of 0.879 and 100% success rate (without distractor: 0.896/100%) using the same amount of samples to plan. Our planner is able to ignore the distractors using our attention structure at every step to only attend to the relevant components in the scene. We also added an example trajectory with distractor dough pieces to our website under "CRS with distractors".

| Planning Horizon | 3 | 4 | 5 | 6 |
|---|---|---|---|---|
| Performance | 0.896 | 0.866 | 0.90 | 0.878 |
| Success Rate | 5/5 | 4/5 | 5/5 | 4/5 |

**Table 11:** PASTA's performance v.s. varying numbers of planning horizon in CRS.

## 5 Further Discussion on Limitations and Future Work

**More Efficient Planning** Planning skill sequences with a large search space is a challenging problem by itself but much progress has been made by the task and motion planning community to obtain a plan skeleton [54, 55, 56]. For example, Caelan et al. [54] propose two methods, the first one is to interleave searching the skill sequence with lower-level optimization and the second one is to have lazy placeholders for some skills. Danny et al. [56] propose to predict skill sequences from visual observation. Recent works have also explored finding skill sequences using pre-trained language models [57, 58].

**Sim2Real Transfer for Real Dough** One possible approach is to train with domain randomization to make the policy more robust to changing dynamics (e.g. stickiness) of dough. Another option is to perform online system identification of the dough dynamics parameters [59, 60] or real2sim methods [61, 62]. In future work, we can also integrate our method with other works that perform low-level dough manipulation in the real world, such as recent work from Qi et al. [63].

**Goal Specification** Our planner requires specifying the goal with a point cloud and coming up with a point cloud goal is not always easy. However, rapid progress are being made with language-conditioned manipulation and future work can combine language to specify more diverse tasks.

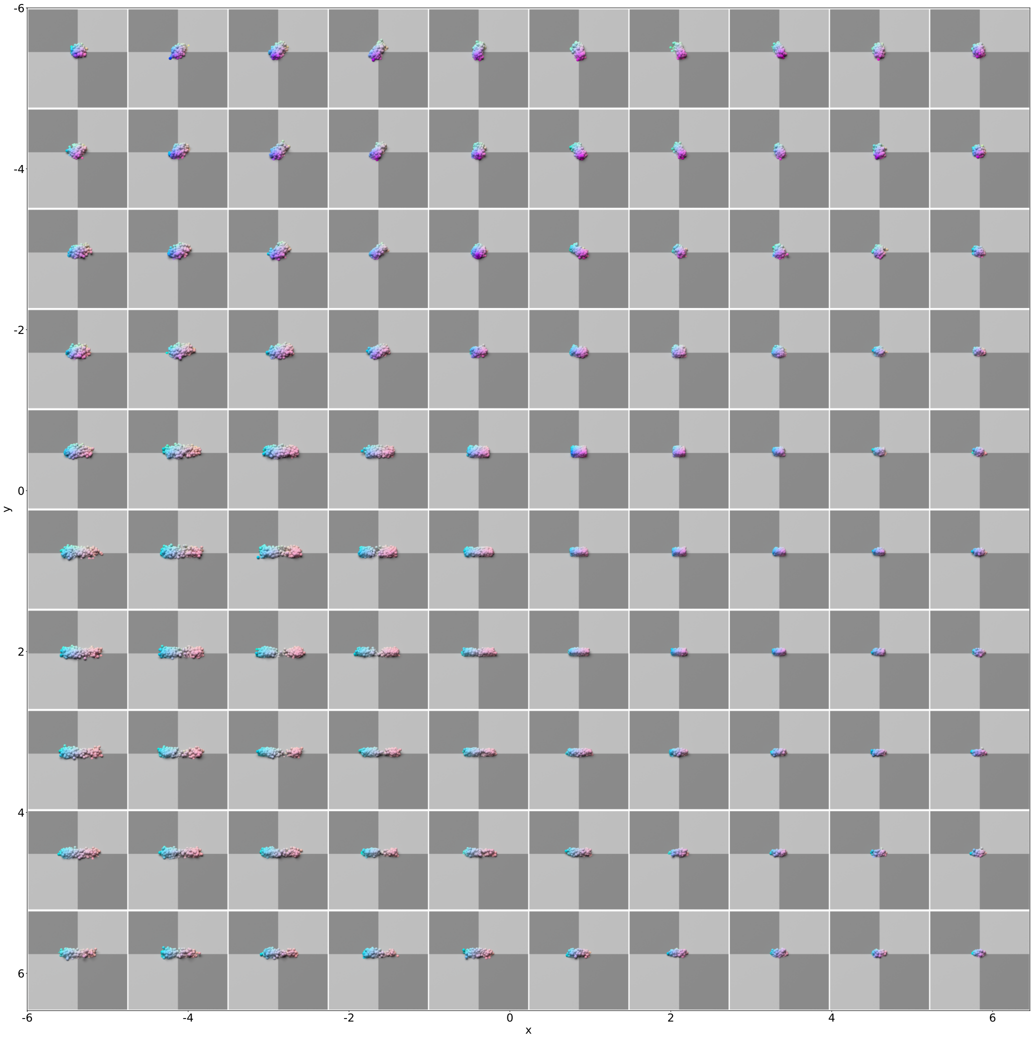


**Figure 6:** Latent space of PASTA in CutRearrange. We sample coordinates on a grid from the 2D latent space encoding the shapes and then decoding each latent vector into a point cloud. We then rearrange the decoded point cloud into the grid based on the corresponding coordinates in the latent space.

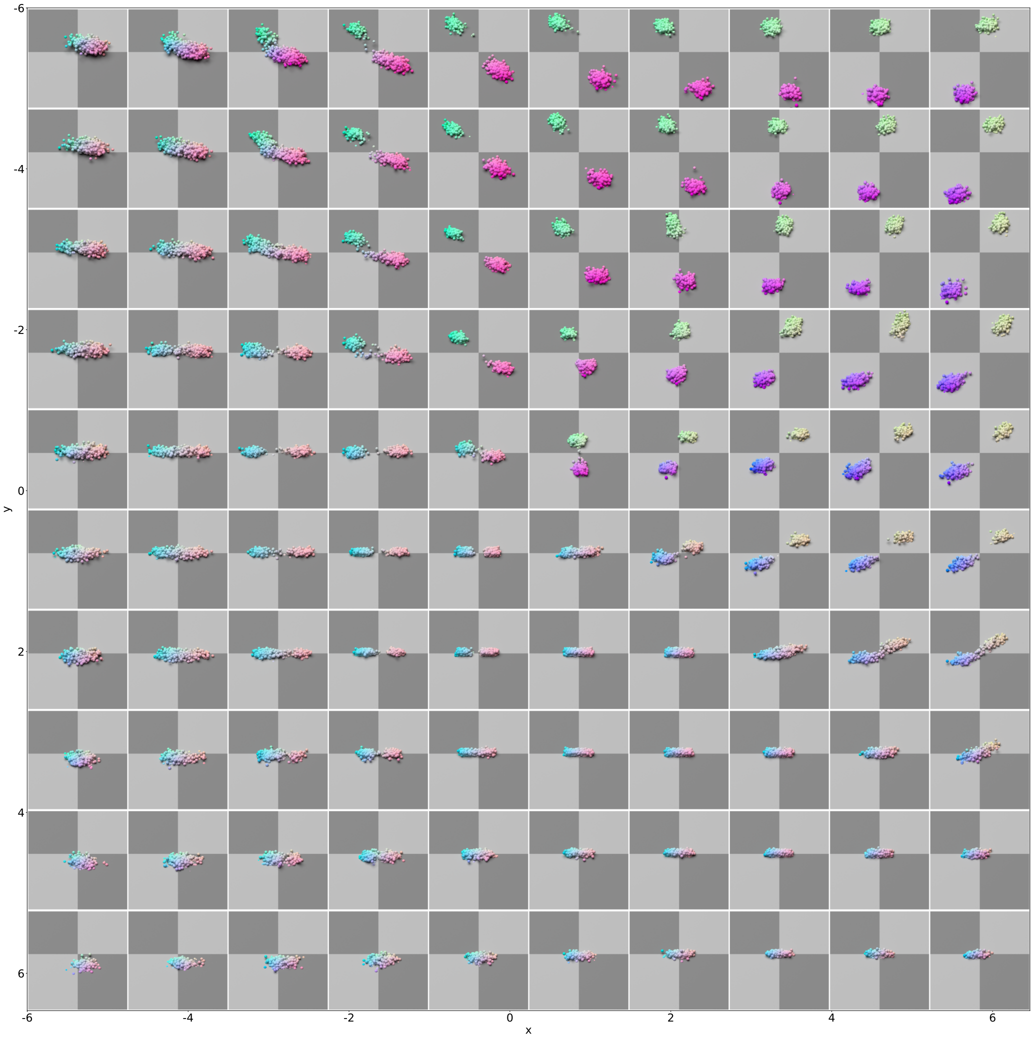


**Figure 7:** Latent space of Flat 3D in CutRearrange. We sample coordinates on a grid from the 2D latent space encoding the shapes and then decoding each latent vector into a point cloud. We then rearrange the decoded point cloud into the grid based on the corresponding coordinates in the latent space.