# DiMSam: *Di*ffusion *M*odels as *Sam*plers for Task and Motion Planning under Partial Observability

Xiaolin Fang[1], Caelan Reed Garrett[2], Clemens Eppner[2],
Tomás Lozano-Pérez[1], Leslie Pack Kaelbling[1], Dieter Fox[2]

***Abstract*— Task and Motion Planning (TAMP) approaches are effective at planning long-horizon autonomous robot manipulation. However, it can be difficult to apply them to domains where the environment and its dynamics are not fully known. We propose to overcome these limitations by leveraging deep generative modeling, specifically diffusion models, to learn constraints and samplers that capture these difficult-to-engineer aspects of the planning model. These learned samplers are composed and combined within a TAMP solver in order to find action parameter values jointly that satisfy the constraints along a plan. To tractably make predictions for unseen objects in the environment, we define these samplers on low-dimensional learned latent embeddings of changing object state. We evaluate our approach in an articulated object manipulation domain and show how the combination of classical TAMP, generative learning, and latent embeddings enables long-horizon constraint-based reasoning. We also apply the learned sampler in the real world. More details are available at https://sites.google.com/view/dimsam-tamp.**

## I. INTRODUCTION

Autonomous robot manipulation in real-world environments is challenging due to large action spaces, long periods of autonomy, the need for contact-rich interaction, and the presence of never-before-seen objects. Although it is in principle possible to learn direct policies for manipulation through imitation or reinforcement learning, these methods generally have particular difficulty as the action space dimensionality and behavior horizon increase. In contrast, Task and Motion Planning (TAMP) [1] approaches have an advantage on long horizon tasks, because they perform model-based reasoning to search over possible futures.

One challenge in TAMP is the search over continuous action parameters. For example, to place an object into a closed microwave, the search must plan an opened configuration for the microwave, a robot trajectory that opens the microwave, a grasp for the object, a placement pose for the object, and the robot's path while moving the object that satisfies collision constraints. To find this plan, we need to be able to test whether these *constraints* are satisfied given the continuous values. These constraints can be prohibitive to engineer when they involve dynamic interactions and partially observed objects. *Our goal is to improve sampling-based TAMP strategies by learning to generate constraint-satisfying samples.*

There has been recent progress in methods for learning deep *generative models*. These models are trained on a distribution $p(X)$ of possibly complicated variables $X$ (such as images, point clouds or trajectories) and then can be queried to produce samples $x \sim p(X)$ at test time. In this paper, we use *diffusion models* [2], [3] to represent distributions over continuous state and action parameters. We deploy them by making conditional sample queries in a sampling-based TAMP solver. These distributional models are critical for maintaining completeness: by predicting *diverse* samples, a TAMP solver is able to backtrack through choices of parameter values to find a globally satisfying set.

[1]MIT CSAIL {xiaolinf,tlp,lpk}@csail.mit.edu
[2]NVIDIA {cgarrett,ceppner,dieterf}@nvidia.com

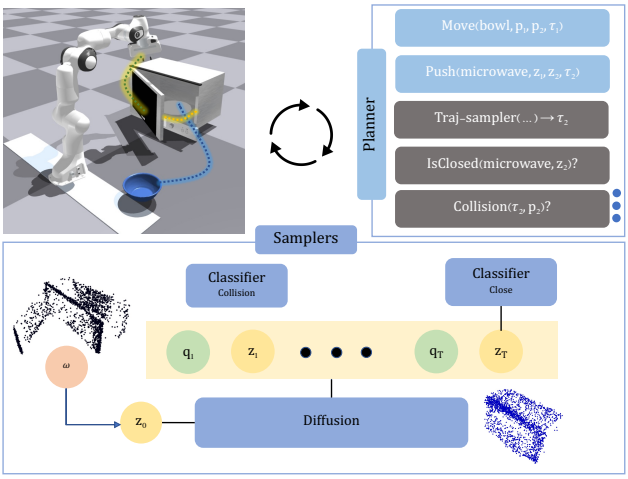



Fig. 1: Task and motion planner searches for feasible plans with learned samplers and classifiers. The bottom shows the constraint graph to push close a microwave door. A diffusion model samples a trajectory of latent microwave states $z_1, ..., z_T$ and robot configurations $q_1, ..., q_T$ that reaches an close state $z_T$. Some links are omitted for simplicity.

Learned samplers are particularly valuable in cases of increased partial observability, where the samplers are conditioned on an image or point cloud, from which it would not be viable for a human to deduce what action parameters would be appropriate. Learned models can enable the solution of a much broader range of problems than traditional engineered samplers.

**The contributions of this work are as follows:**

1) We use diffusion models as a generative representation of TAMP constraints in the form of samplers.
2) We define these constraints on a latent embedding of object state, allowing them to be applied to previously unseen objects with no known models.
3) These latent embeddings enable us to perform TAMP in a hybrid latent-engineered state space.

4) We showcase our approach on articulated-object manipulation tasks that require multi-step reasoning.

## II. RELATED WORK

We build on prior work in TAMP and generative modeling. TAMP considers planning in *hybrid* spaces where there are both continuous and discrete state and action parameters[4], [1], [5]. Traditionally, these models are specified by a human; however, increasingly, aspects of these models, such as constraints, samplers, and parametric operators, have been learned. Wang *et al.* [6] and Kim *et al.* [7] use Gaussian Processes and GANs to learn low level samplers for TAMP. Silver *et al.* [8] learn parameterized skills along with parametric operators that model these skills and samplers for continuous parameters, assuming observability. Mao *et al.* [9] learn constraints and planning heuristics that operate directly on images but assume a set of parameterized skill policies. There is also related work on learning for other manipulation planning frameworks [10], [11], [12], but usually not considering complicated objects such as articulated objects.

Recent work uses generative models for decision making [13], [14]. There is also a line of work doing trajectory modeling using auto-regressive transformer models [15], [16] or masked autoencoders [17]. Diffuser [13] uses a diffusion model [2], [3] for offline reinforcement learning (RL) and a manipulation problem. Diffusion Policy [18] uses diffusion models to represent visuomotor policies and shows better performance than IBC and LSTM-GMM, while also being more stable in training. In this paper, we apply a similar model but in the context of learning samplers for TAMP. Our focus is to use the learned model *combinatorially* with other samplers to solve multi-step planning tasks.

## III. SAMPLING-BASED TAMP

We are interested in using generative learning to extend the applicability of TAMP to partially observed settings where classic engineering approaches struggle.

### A. TAMP Problem Description

A generic TAMP problem $\Pi = \langle \mathcal{S}, \mathcal{A}, s_0, S_* \rangle$ can be described by a state-space $\mathcal{S}$, a set of parameterized actions $\mathcal{A}$, an initial state $s_0 \in \mathcal{S}$ and a set of goal states $S_* \subseteq \mathcal{S}$. States are comprised of a set of hybrid variables with values that can change over time. Each parameterized action $a \in \mathcal{A}$ takes in a tuple of hybrid parameters $x$ that instantiate the *preconditions* and *effects* of the action. The preconditions dictate variable values that enable *action instance* $a(x)$ to be correctly executed in state $s$. The effects specify changes to the variable values in state $s$ that give rise to subsequent state $s'$ after executing action instance $a(x)$. Critically, in order for an action instance $a(x)$ to be valid, its parameters $x$ must satisfy a conjunctive set of *constraints* $a.\textbf{con} = \{c_1, ..., c_n\}$, namely $\bigwedge_{i=1}^{n} [c_i(x) = \textbf{True}]$.

The objective of planning is to find a *plan* $\pi$, a finite sequence of action instances $\pi = [a_1(x_1), ..., a_k(x_k)]$, that when executed from state $s_0$, produces a state $s_* \in S_*$. Actions in a plan often share parameters due to variables persisting in the state. Thus, the parameters across a valid plan $\bigcup_{i=1}^{k} x_i$ must jointly satisfy the corresponding set of action constraints $C_\pi = \bigcup_{i=1}^{k} a_i.\textbf{con}$. Solving a TAMP problem requires simultaneously identifying a sequence of parameterized actions $[a_1, ..., a_k]$ along with parameter values $[x_1, ..., x_k]$ that satisfy their constraints.

### B. Conditional Samplers

Given set of constraints $C$, solving for parameter values that satisfy them is a Hybrid Constraint Satisfaction Problem (H-CSP) [1]. These problems are addressed using joint *optimization* [4] and individual *sampling* [19] techniques. Joint optimization methods typically treat the ensemble of parameters and constraints as a single mathematical program and solve for satisfying values all at once.

In contrast, individual sampling techniques leverage *compositionality* through conditional sampling, where the outputs of a sampler for one constraint, *e.g.*, a stable grasp and resting pose for an object become the inputs to another, *e.g.*, an inverse kinematics (IK) solver for the robot. A *conditional sampler* for a constraint $c$ with $m$ arguments is a function from $\alpha$, a tuple of $k < m$ argument values to a generator of tuples $\beta_i$, each of which is length $m-k$ and has the property that when its values are properly interleaved with the values $\alpha$, the resulting length-$m$ tuple of values $x_i$ satisfies $c$, (*i.e.* $c(\alpha, \beta) = \textbf{True}$). We seek samplers that are:

- **Sound:** they only generate values that satisfy the constraint they represent.
- **Diverse:** they generate multiple diverse values that can be filtered with other constraints via rejection sampling.
- **Compositional:** they can be combined with others to produce joint samples that satisfy multiple constraints.

In the articulated object manipulation domain described in Sec. I, we require samplers that 1) perform inverse kinematics, 2) generate stable placement and grasp poses, 3) check collisions, 4) generate desirable door states, and 5) model the contact dynamics of doors. When the world is known, samplers 1-4 can be readily engineered through geometric reasoning. However, engineering sampler 5 accurately is non-trivial due to the contact-rich nature of the interaction. Moreover, when the world is partially observable and we do not *a priori* know the geometry of objects, engineering samplers 1-4 themselves is challenging.

This motivates our approach, where we use diffusion models to learn conditional samplers that represent generative models of action constraints. These learned samplers can be incorporated in any sampling-based TAMP system [20]. Additionally, we show that learned samplers can be applied to parameters that describe the latent state of unknown objects, and allow the planner to search for plans in the latent space. In TAMP frameworks that use symbolic predicates as state abstractions, predicate definitions, such as whether a microwave is 'open' or 'closed', are hard to hand-engineer especially with unknown objects. A learned classifier can be used instead, to bias the diffusion sampling in drawing conditional samples that make the predicate **True**(see Sec. IV-C for details).

## IV. Diffusion Models as Learned Samplers

We seek to learn samplers that generate samples $x$ that satisfy constraint $c$. We require a *training dataset* $\mathcal{D}_c = \{x_1, ..., x_N\}$ of $N$ length-$m$ parameter tuples $x_i$ that satisfy constraint $c$, *i.e.* $c(x_i) =$ **True**. Then, we learn an implicit probability distribution $p(x)$ over parameters $x$ that satisfy constraint $c$ using dataset $\mathcal{D}_c$. Finally, through incorporating condition terms in the sampling process, we turn the unconditional model $p(x)$ into conditional models $p(\beta \mid \alpha)$ that become the basis for conditional constraint samplers.

### A. Generative Models

We propose to use deep *generative models*, specifically diffusion models, as samplers. A generative model flexibly captures a distribution and diffusion models are shown to be good at capturing multi-modality in the presented data [18]. One can draw both *unconditional* samples from this distribution and *conditional* samples that satisfy a constraint. In planning, a diverse set of unconditional samples can be used for rejection sampling, to jointly satisfy a set of nonhomogeneous constraints, but if there is a given condition for the target, drawing conditional samples directly may be more efficient.

### B. Diffusion Models

Diffusion models [2], [3] are a class of deep generative models. They produce samples $x^{(0)}$ from a learned distribution $p(x)$ by iteratively applying a denoising procedure $p(x^{(t-1)}|x^{(t)})$, starting from Gaussian noise $x^{(T)}$. The denoising procedure makes transitions according to

$$p(x^{(t-1)}|x^{(t)}) := \mathcal{N}(x^{(t-1)}; \mu_\theta(x^{(t)}, t), \Sigma_\theta(x^{(t)}, t)). \quad (1)$$

Here, $\mu_\theta$ and $\Sigma_\theta$ are time-conditional functions with learnable parameter $\theta$. During training, a forward process will gradually add random noise to the original data point $x^{(0)}$. The network is trained to predict the noise added on a data point to generate the corrupted data point. Once the network is trained, the model can be used to draw samples from $p(x)$ starting from a Gaussian noise sample, according to Eq. 1.

### C. Conditional Diffusion Sampling

Our key use case for generative models is conditional sampling with one or more constraints. When conditionally sampling a diffusion model, one can use *classifier-based* [21] or *classifier-free* [22] guidance. Classifier-based guidance uses the gradient of a classifier to bias the sampling of an unconditional diffusion model. In contrast, the classifier-free guidance doesn't require another model but assumes knowledge of all conditions at training time.

We hope to achieve compositional generality, by allowing the model to work on new tasks when given new constraints. Namely, we don't assume that we will know all potential condition types when training models but would like to allow new models to be trained afterward and work with the existing models if a new constraint comes up. Thus, we opt to use *classifier-based* guidance as it allows us to combine the unconditional diffusion model with other new models after it is trained.

For classifier guidance, as shown by Dhariwal *et al.* [21], the denoising procedure can be approximated as

$$p(x^{(t-1)}|x^{(t)}) \approx \mathcal{N}(x^{(t-1)}; \mu_\theta + \Sigma_\theta g_\phi, \Sigma_\theta), \quad (2)$$

where $g_\phi = \nabla_{x^{(t)}} \log(p_\phi(y|x^{(t)}))$ is the gradient from a classifier $p_\phi(\cdot)$, that models the likelihood of sample $x^{(t)}$ having property $y$, to bias the sampling. This sampling process can be significantly more efficient than rejection sampling if one's goal is to get $x^{(0)}$ that has property $y$.

### D. Latent Parameter Encoding

In our TAMP domain, we are interested in learning samplers that operate on objects that have never seen before, without access to a model of their shape and kinematics. We can only sense them through observations $\omega$ in the form of segmented partial point clouds, projected from depth images. Moreover, for articulated objects, the geometry can non-rigidly change over time. Modeling the explicit dynamics and transitions of such changes in the space of point cloud is challenging and inefficient.

For a more compact encoding and for making the constraint learning problem easier, we train a point cloud encoder $\phi_{enc}$ (and decoder $\phi_{dec}$) to compress observations of objects $\omega$ into latent state statistics $z$. Observed partial point clouds $\omega \in \mathcal{R}^{N\times 3}$ are encoded as a latent vector $z \in \mathcal{R}^{d_z}$, where $d_z$ is the dimensionality of the latent vector. The weights of the encoder and decoder are frozen after training, and only the encoder is required at planning time.

### E. Examples of Learned Samplers and Classifiers

In the following examples, we use variables $o$ for object type information, $z$ for latent object shape representations, $q$ for robot configurations, $g$ for a grasp transform, and $p$ the pose of objects of a known shape.

*1) Grasp Sampler:* One common sampler is a grasp pose sampler for a stable grasp constraint `LearnedGrasp` [23]. This constraint $p(x) = p(o, g)$ models the set of end-effector grasp poses $g$ on an object $o$ that have a high probability of stability. Here, object $o$ is represented by a category and its segment in an observed point cloud $\omega$. The sampler for this constraint $p(g \mid o)$ conditions on the object. This sampler can be used together with classifiers for conditional sampling. If there is a specific constraint on a class object, *e.g.* grasp on the handle for all mugs, a lightweight classifier $p_\phi(o, g)$ can be trained to bias the sampling of $p(g \mid o)$.

*2) Push Sampler:* A more complex constraint and sampler that we wish to learn models a robot interacting with an articulated object via making contact, for example, by pushing. The `DiffusionPush` constraint governs the evolution of the robot's target configuration $q$ and a latent representation of the state $z$ of object $o$ over $T$ time steps:

$$p(x) = p(o, z_1, q_1, z_2, q_2, ..., z_T, q_T). \quad (3)$$

For such trajectory-level models, conditioning can be added to different time steps in multiple ways

to produce an ensemble of samplers, including a *forward* sampler $p(q_1, z_2, q_2, ..., z_T, q_T | o, z_1)$ that draws future state from a start latent state $z_1$, a *backward* sampler $p(z_1, q_1, z_2, q_2, ..., q_T | o, z_T)$ that infers *pre-image* of latent state $z_T$, and a *two-point* sampler $p(q_1, z_2, q_2, ..., q_T | o, z_1, z_T)$ that fills possible transition between two latent states $z_1, z_T$.

*3) Object State Classifiers:* Often, the goal conditions that define $S_*$ require specific objects to be at a state that is semantically meaningful for a human, for example, a state that a human considers open `ClassifyOpen[·]` or closed `ClassifyClosed[·]`. To model this, we learn a classifier $p_\phi(o, z)$ on the object $o$ and the latent state $z$. These classifiers can condition a push sampler to generate reachable latent states $z_T$ that satisfy the classifier, namely $p(q_1, z_2, q_2, ..., z_T, q_T \mid o, z_1)$ and $p_\phi(o, z_T)$, as shown in Fig.1.

*4) Collision Classifier:* Similarly, trajectory samples must not collide with other objects, for example, a door being opened should not collide with another object. To model this, we learn a classifier $p_\phi(o, z, o_2, p_2)$ for constraint `ClassifyCollision` on the object $o$ and its latent state $z$ versus another object $o_2$ and its relative pose $p_2$.

## V. Implementation

We now ground our general approach of TAMP using diffusion models as samplers in a concrete domain, as shown in Fig. 1. We consider a robot manipulating a microwave to achieve goals, by planning using a set of learned models.

### A. TAMP Formulation

We instantiate our TAMP problems $\Pi$ using PDDLStream [19], an extension of Planning Domain Definition Language (PDDL) [24] that supports planning with continuous values using sampling operations. Planning state variables and action constraints are represented using *predicates*, Boolean functions with zero or more parameters.

The set of goal states $S_*$ is described by a logical formula over predicate atoms. For example a goal for the microwave $o_m$ to be open $\texttt{Open}(o_m)$ is defined by:

$$\texttt{Open}(o_m) \equiv \exists\, z.\, \texttt{AtLatent}(o_m, z) \wedge \underline{\texttt{ClassifyOpen}(o_m, z)},$$

where the predicate `ClassifyOpen` is a learned classifier. Parameterized actions are defined by their 1) name, 2) parameters, 3) constraints (**con**) that valid parameter values satisfy, 4) preconditions (**pre**) that hold to prior to executing the action, and 5) effects (**eff**) that modify the state.

### B. Actions with Learned Constraints

The `push` action involves the major learned constraints in this domain (Figure 2). Its parameters are an articulated object $o$ and a sequence of latent states $z_1, ..., z_T$ and robot configurations $q_1, ..., q_T$. After applying the action, the robot moved from configuration $q_1 \to q_T$ and object $o$ moved from latent state $z_1 \to z_T$. The key constraint is `DiffusionPush`, which is learned via a diffusion model as described in Sec. IV-E. This constraint can be sampled in several orientations depending on the fixed parameter values passed and the other constraints, such as the aforementioned forward sampler and two-point sampler.

$\texttt{push}(o, z_1, q_1, ..., z_T, q_T)$
- **con:** $[\underline{\texttt{DiffusionPush}(o, z_1, q_1, ..., z_T, q_T)},$
  $\underline{\neg\texttt{Unsafe}(o, z_1)}, \ ..., \ \underline{\neg\texttt{Unsafe}(o, z_T)}]$
- **pre:** $[\texttt{AtLatent}(o, z_1), \texttt{AtConf}(q_1), \texttt{Empty}()]$
- **eff:** $[\texttt{AtLatent}(o, z_T), \texttt{AtConf}(q_T),$
  $\neg\texttt{AtLatent}(o, z_1), \neg\texttt{AtConf}(q_1)]$

Fig. 2: The `push` action description.

`Unsafe`$(o, z)$ imposes the collision-free constraint on the whole motion, which is evaluated by a learned classifier `ClassifyCollision` that directly operates on the latent state $z$ of object $o$ and $p_2$, the relative poses of $o_2$ to $o$.

$$\texttt{Unsafe}(o, z) \equiv \exists\, o_2, p_2.\, \texttt{AtPlace}(o_2, p_2) \wedge \underline{\texttt{ClassifyCollision}(o, z, o_2, p_2)}.$$

Consider a problem where the microwave is initially closed, but the goal is for the block to be in the microwave. The TAMP system needs to infer that it should push the microwave sufficiently open so that it can pick and stow the block. If there is obstacle, it also needs to infer that the obstacle should be moved away before opening the door. Below is an example plan structure. Values in bold are fixed as they are initial state given to the system.

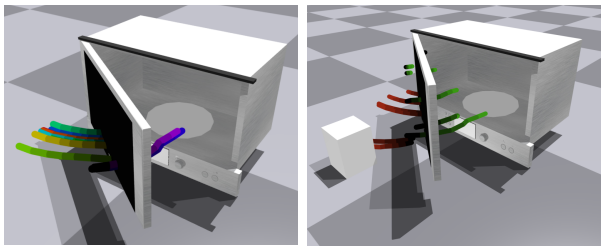
$$\begin{aligned}\pi = [&\texttt{move}(\mathbf{q_0}, \tau_1, q_1), \texttt{push}(o_m, \mathbf{z_0}, q_1, ..., z_1, q_2),\\ &\texttt{move}(q_2, \tau_2, q_3), \texttt{pick}(\texttt{block}, g, \mathbf{p_0}, q_3),\\ &\texttt{move}(q_3, \tau_3, q_4), \texttt{place}(\texttt{block}, g, p_*, q_4)]\end{aligned}$$

### C. Environment and Data Collection

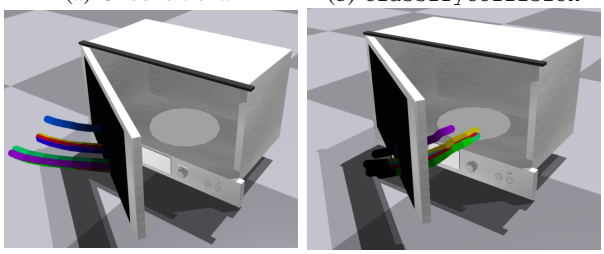
We perform our experiments in the IsaacGym [25] physics simulator. To learn the data distribution that satisfies `DiffusionPush` constraint, we generate a training set $\mathcal{D}$ of push action instances by simulating manipulation trajectories in IsaacGym. The manipulation trajectories are generated according to a custom policy that leverages the ground truth state, with known object models and contact forces. Action is pushing perpendicular to segmented door surface. We use 10 microwave assets from PartNet [26] for training.

During data collection, we treat the robot as a disembodied gripper, so the robot's configuration is an end-effector position $q \in \mathbb{R}^3$. This representation yields a simple parameterization of the skill and allows the learned model be flexibly integrated into a full arm motion using inverse kinematics and motion planner in a TAMP framework.

Partial point cloud observations are generated from depth cameras with randomized viewpoints. There are 101 valid trajectories in total, which are then randomly clipped into segments for training.



(a) Unconditional (b) `ClassifyCollision`



(c) `ClassifyOpen` (d) `ClassifyClose`

Fig. 3: Samples from the `DiffusionPush` model, colored from darker to brighter from start to end of the trajectories. (a) No extra condition except for $z_0$. (b) Rejection sampling by `ClassifyCollision`. Rejected samples are colored red. (c) (d) Classifier-guided conditional sampling on `ClassifyOpen` and `ClassifyClose`.

### D. Training Details

We collected 4746 point clouds, from 101 trajectories in total. We use PointNet++ [27] as the point cloud encoder network $\phi_{enc}$ and use the same decoder $\phi_{dec}$ as Cai *et al.* [28]. The latent vector size $d_z$ is 256. The network is trained with a batch size of 196. All other hyper-parameters are the same as in Cai *et al.* [28]. After the point cloud encoder is trained, the weights are frozen and used as the encoder when training all other models.

For the trajectory diffusion model, we use the UNet as in Diffuser [13]. The number of diffusion steps is 250, the trajectory length $T = 8$, and the batch size is 32. Microwave state (open/close) classifiers are 3-layer MLPs. The inputs to the collision network are point cloud encoding, obstacle location, and size approximated as bounding boxes.

## VI. Experiments

We experiment on three settings within our domain:

1) Manipulating to a specific state (Sec. VI-A);
2) Manipulating to an abstract set of states (Sec. VI-B);
3) Joint searching and sampling (Sec. VI-C).

### A. Planning to specific states

We first test whether the learned model can generate high-quality conditional samples. The initial and goal microwave door configurations are randomly sampled. Point clouds captured from a randomized viewpoint are added as the goal condition for the first and last time step $\omega_1, \omega_T$ of diffusion sampling. They are encoded into latent embeddings using learned point cloud encoder $z_1 = \phi_{enc}(\omega_1)$ and $z_T = \phi_{enc}(\omega_T)$. Then, we use them as conditions to sample the rest of the trajectory $\langle q_1, z_2, q_2, ..., z_{T-1}, q_{T-1} \rangle$ from the diffusion model. This corresponds to the two-point sampler in section IV-E.2. The sampled action waypoints $q_1, ..., q_{T-1}$ are then executed in the simulator.

We compared the performance of our model to two baselines: 1) a *discriminative model* that directly regresses from a pair of start and end positions $z_1, z_T$ to $q_1, ..., q_{T-1}$ and 2) an energy-based model (EBM).

| Method | Avg. Error (std) |
|---|---|
| Regression model | 0.127 (0.214) |
| EBM | 0.191 (0.208) |
| **Diffusion** | **0.094 (0.091)** |

TABLE I: Planning error given specific goal states.

Table I shows the average (and standard deviation) of the absolute distance between the target angle and the actual ending angle on 100 testing trajectories (in radian). The diffusion model has the lowest error. We found the diffusion model to be easier to train and to have a more stable gradient during training, when compared to the EBM, which is also observed in [18]. The regression model also trained stably. However, since the regression model is deterministic, given the same point cloud embeddings $z_1, z_T$, the network's predictions will be identical. The diffusion model can generate more diverse samples that satisfy the given constraints, which can improve system robustness if used as a candidate trajectory sampler. See Figure 3 for an illustration.

### B. Planning to reach goal sets

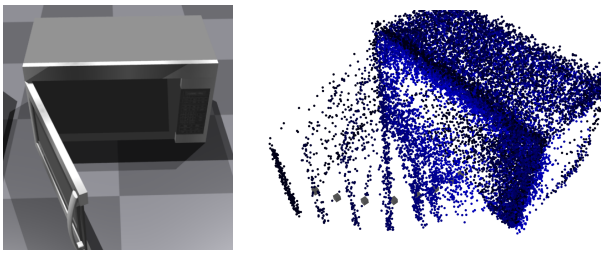
Goals, consisting of sets of possible satisfying states, can be specified in the form of named classifiers We evaluate the learned model with classifier-guided sampling. Three classifiers are trained to model the point cloud as being "fully closed", "fully open", and "open". The corresponding semantics are door angle $<$0.2, $>$1.4, and $>$1.2 radians. Trajectory samples are drawn from the learned model with these constraints enforced at the end of the trajectory $p_{\phi_k}(z_T) = 1$, according to the transition described in Eq. 2. We evaluate on 100 randomized initial configurations of the microwave, and report the success rate of the actual state satisfying the semantic constraint, after executing the sampled trajectory. Qualitative results are shown in Fig. 3, with the door being fully closed and fully opened. Success rate is reported in Table II. Figure 4 shows the entire sampled trajectory with classifier guidance of 'fully closed', action waypoints as gray boxes.

| Goal Condition | fully closed | fully open | open |
|---|---|---|---|
| Success Rate | 0.94 | 0.92 | 0.90 |

TABLE II: Achieving abstract goals with classifier guidance.

### C. TAMP: Joint searching and sampling

In prior experiments, we demonstrate the quality of learned model in a stand-alone context, where the end-effector is simplified as a moving gripper. In the following experiments, we test the learned models in a complete TAMP system.



(a) (b)

Fig. 4: (a) Actual configuration and (b) sampled state and action trajectory. Sampled latent state $z$ are decoded into point cloud for visualization.

Diffusion models are combined with other samplers such as IK and motion planners, in order to solve a multi-step task.

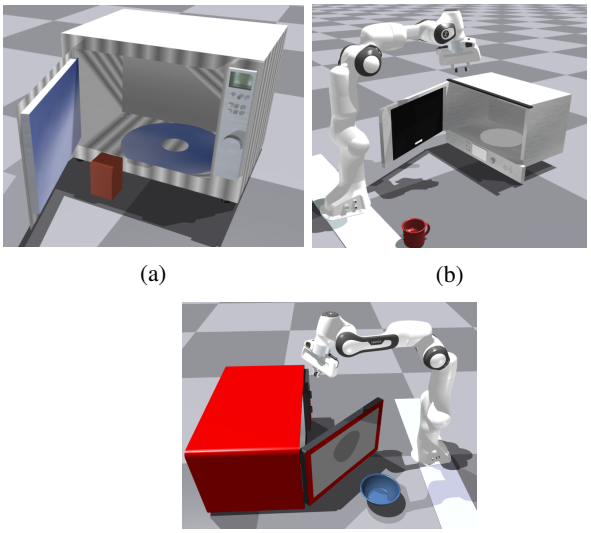
We evaluate on three tasks, which are shown in Fig. 5. 1) *Close* Goal state is the microwave door at closed location. Initial state is the door at an opened position, where there is an obstacle blocking in between the door and the base. 2) *Stow-close* The goal is to have an object stowed in the microwave and door closed at the end. The object is initialized to be at a fixed location that won't block the microwave. 3) *Stow-close-blocked* The goal is the same as *Stow-close*. However, the door needs to be fully opened before stowing as the object is larger. The object is initialized near the microwave which may block the opening action. More details on the domains are available here.

As in the previous experiments, for the microwave, the planner only receives a partial point cloud captured from a randomized viewpoint. The initial location and geometry of the object to be stowed and obstacle are known. We use a Franka Emika robot with a mobile base link along the x-axis.

In addition to learned classifiers and samplers, we use the robot URDF and computed IK solutions for the whole arm movement. The predicted waypoint is set as the target end point of gripper. For collision checking between robot arm and a predicted microwave state, we approximate the arm using bounding boxes and query the learned collision checker. For the initial observation, collision checking is based explicitly on closest distance to observed points. Motion plans are computed by bidirectional RRT.

We use PDDLStream to jointly search the structure of the plan and trajectory sampling. The *Adaptive* algorithm with a *search-sample* ratio of 1/15 is used for all tasks. We use weighted A$^*$ and fast-forward heuristic. Planning and execution success rates are shown in Table III. Each task is evaluated with 15 runs. Average planning time and plan length are reported based on solved runs.

**Behavior Analysis** In the first task, the planner will start from the shortest plan of sampling a trajectory to close the door and check whether that is collision-free with the obstacle. The collision checker uses the predicted state of the microwave from the sampled trajectory, $z_2, ..., z_T$. As the returned results indicate collision, the planner will keep searching for alternative plans. The most common plan is `[move(`$o_1$`), push(`$o_m$`)]`. In *Stow-close*, note that the goal `Closed(`$o_m$`)` $\wedge$ `Stowed(`$o_1$`)` is not ordered, it is possible for the planner to find plan like `[push(`$o_m$`), stow(`$o_1$`)]`. However, such plan will result in a collision between predicted microwave state and the stowing object, and get rejected. The third task is the most challenging one. A precondition of door being opened is added to the domain. Due to the combinatorial complexity, the search space grows exponentially larger compared to the previous two tasks. So it also takes a longer time to solve, with two runs hit the timeout (200 seconds). Other than the challenges in searching in a large space, a common failure mode is for the classifier to mis-estimate the open/closed state of the door. We also observe execution failures when the sampled trajectory is too close to the axis of the door and physically infeasible. Incorporating uncertainty or effort estimates from learned samplers can be an interesting future direction.



(a) (b) (c)

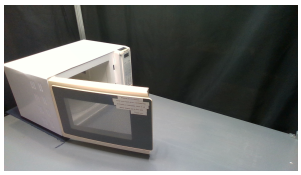
Fig. 5: (a) *Close*, (b) *Stow-close*, (c) *Stow-close-block*.

| Task | Solved | Length | Time | Achieved |
|---|---|---|---|---|
| Close | 1.00 | 1.93 | 29.41 | 0.67 |
| Close$^{c*}$ | 1.00 | 1.93 | 15.71 | 0.87 |
| Stow-close | 0.93 | 1.93 | 14.51 | 0.71 |
| Stow-close$^{c*}$ | 1.00 | 2.00 | 10.27 | 1.00 |
| Stow-close-B$^{c*}$ | 0.87 | 4.00 | 77.70 | 1.00 |

TABLE III: Results on joint planning using PDDLStream. 'Solved' indicates the planning success rate. 'Achieved' indicates execution success rate. Task names with superscript($^{c*}$) are using conditional samplers.

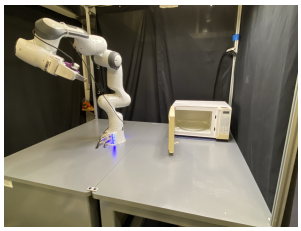
**Conditional vs. rejection sampling** As we mentioned in previous sections, conditional sampling is more efficient in drawing samples with given constraint. Classifier-guided diffusion sampling require backpropagating the gradient through the classifier network, which makes it more expensive to call. However in our experiments, the efficiency gain from searching outweighs the additional cost and shows the advantage to rejection sampling. As shown in Table. III, tests with condtional samplers ($\cdot^{c*}$) take fewer time to solve.

It is important to note that this is a problem that cannot be addressed by classical TAMP methods, because the kine-

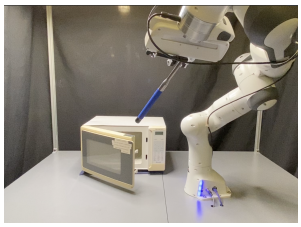


(a) Observed image (not used in model).

(b) Observed point cloud, top-down view.



(c)



(d)

Fig. 6: Real world setup.

matic and shape models of the microwave are unknown. In addition, note that no new learning was required to do this problem—once the individual generative models are trained for each constraint, they can be combinatorially recombined to solve a wide variety of problems. This is in contrast to direct policy learning methods, which require training on new tasks and generally struggle with long horizons unless given a carefully crafted reward function.

## VII. Real world Deployment

We apply the model trained in simulation directly to the real world, without finetuning, on door closing and opening tasks.

We capture the depth image from a RealSense D435 depth camera mounted on the Franka gripper. The point cloud is segmented and sent to the learned `DiffusionPush` model. In the door closing and opening tasks, we use conditional sampling based on `ClassifyClose` and `ClassifyOpen` classifiers trained in simulation. The sampled trajectories are further rejected by IK and motion sampler, with collision checking to the observed partial point cloud.

In the door-opening task, due to the small size of the microwave, the Franka robot gripper can't fit in between the microwave door and the microwave base. To fix that, we let the robot hold a stick. The predicted waypoints are set as the target location of the stick endpoint. Since our model only predicts the end-effector waypoint, we can make such changes by modifying the IK and motion planner to take into consideration the stick easily, without changing the trained model. It shows the flexibility of using a simplified encoding of action.

The observed image and point cloud are shown in Fig. 6. The RGB image is not used by the model. Due to the reflective material of the microwave and the glass used in microwave door, there is a lot of missing depth readings in the depth image. Despite the severe partial observability, our model is still able to generate target samples. Videos are available at the website.

## VIII. Conclusion

We apply diffusion models for sampler learning in TAMP and show that conditional sampling on learned models can be used to draw samples that satisfy constraints in a TAMP problem. We instantiate an example of such samplers in a concrete articulation manipulation domain, in learning pushing constraints, and hope this strategy can be applied in broader domains.

## References


[1] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated Task and Motion Planning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, 2021.

[2] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International Conference on Machine Learning*. PMLR, 2015, pp. 2256–2265.

[3] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[4] M. Toussaint, "Logic-geometric programming: an optimization-based approach to combined task and motion planning," in *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 2015, pp. 1930–1936.

[5] A. Curtis, X. Fang, L. P. Kaelbling, T. Lozano-Pérez, and C. R. Garrett, "Long-horizon manipulation of unknown objects via task and motion planning with estimated affordances," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1940–1946.

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

[7] T. L.-P. Beomjoon Kim, Leslie Pack Kaelbling, "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)*, 2018.

[8] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling, "Learning neuro-symbolic skills for bilevel planning," in *6th Annual Conference on Robot Learning*.

[9] J. Mao, T. Lozano-Pérez, J. B. Tenenbaum, and L. P. Kaelbling, "Pdsketch: Integrated domain programming, learning, and planning," in *Advances in Neural Information Processing Systems*.

[10] A. H. Qureshi, A. Mousavian, C. Paxton, M. C. Yip, and D. Fox, "Nerp: Neural rearrangement planning for unknown objects," *CoRR*, vol. abs/2106.01352, 2021.

[11] D. Driess, J.-S. Ha, M. Toussaint, and R. Tedrake, "Learning models as functionals of signed-distance fields for manipulation planning," in *Conference on Robot Learning*. PMLR, 2022, pp. 245–255.

[12] D. Xu, A. Mandlekar, R. Martín-Martín, Y. Zhu, S. Savarese, and L. Fei-Fei, "Deep affordance foresight: Planning through what can be done in the future," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6206–6213.

[13] M. Janner, Y. Du, J. Tenenbaum, and S. Levine, "Planning with diffusion for flexible behavior synthesis," in *International Conference on Machine Learning*, 2022.

[14] A. Ajay, Y. Du, A. Gupta, J. Tenenbaum, T. Jaakkola, and P. Agrawal, "Is conditional generative modeling all you need for decision-making?" *arXiv preprint arXiv:2211.15657*, 2022.

[15] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," in *Advances in Neural Information Processing Systems*, 2021.

[16] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *arXiv preprint arXiv:2106.01345*, 2021.

[17] F. Liu, H. Liu, A. Grover, and P. Abbeel, "Masked autoencoding for scalable and generalizable decision making," *arXiv preprint arXiv:2211.12740*, 2022.
[18] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song, "Diffusion policy: Visuomotor policy learning via action diffusion," in *Proceedings of Robotics: Science and Systems (RSS)*, 2023.
[19] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "PDDLStream: Integrating Symbolic Planners and Blackbox Samplers," in *ICAPS*, 2020.
[20] C. R. Garrett, C. Paxton, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox, "Online replanning in belief space for partially observable task and motion problems," in *ICRA*. IEEE, 2020.
[21] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Advances in Neural Information Processing Systems*, vol. 34, pp. 8780–8794, 2021.
[22] J. Ho and T. Salimans, "Classifier-free diffusion guidance," *arXiv preprint arXiv:2207.12598*, 2022.
[23] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox, "Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 13 438–13 444.
[24] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL: The Planning Domain Definition Language," Yale Center for Computational Vision and Control, Tech. Rep., 1998.
[25] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021.
[26] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su, "Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 909–918.
[27] 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*, vol. 30, 2017.
[28] R. Cai, G. Yang, H. Averbuch-Elor, Z. Hao, S. Belongie, N. Snavely, and B. Hariharan, "Learning gradient fields for shape generation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.