2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

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*— Generative models such as diffusion models, excel at capturing high-dimensional distributions with diverse input modalities, e.g. robot trajectories, but are less effective at multi-step constraint reasoning. Task and Motion Planning (TAMP) approaches are suited for planning multi-step 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 composing diffusion models using a TAMP system. We use the learned components for constraints and samplers that are difficult to engineer in the planning model, and use a TAMP solver to search for the task plan with constraint-satisfying action parameter values. To tractably make predictions for unseen objects in the environment, we define the learned samplers and TAMP operators on learned latent embedding of changing object states. We evaluate our approach in a simulated articulated object manipulation domain and show how the combination of classical TAMP, generative modeling, and latent embedding enables multi-step constraint-based reasoning. We also apply the learned sampler in the real world. Website: 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. Nonetheless, we have seen prominent progress in generative modeling, that demonstrates its suitability for learning relatively short-horizon robot action distributions. 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. In this work, our goal is to compose learned models using a TAMP framework for solving multi-step robot manipulation problems.

Solving a long-horizon robot planning problem involves determining an appropriate sequence of action instances (the type of action and the objects it is applied to), and finding the values of the continuous-valued parameters for each of the actions. This process can be characterized as constructing and solving a constraint-satisfaction problem (CSP). The high level task plan implies a set of constraints, which defines a CSP. For example, a task plan involving opening the microwave and then stowing an object implies a constraint that the microwave must be sufficiently opened to stow the object. The planner often needs to alternate between the process of constructing and solving CSPs, as some of the CSPs constructed may be unsatisfiable, such as when the robot seeks an inverse kinematics (IK) solution to an unreachable pose. To effectively address this problem, we need to reason about constraints and efficiently find values to solve the CSP.

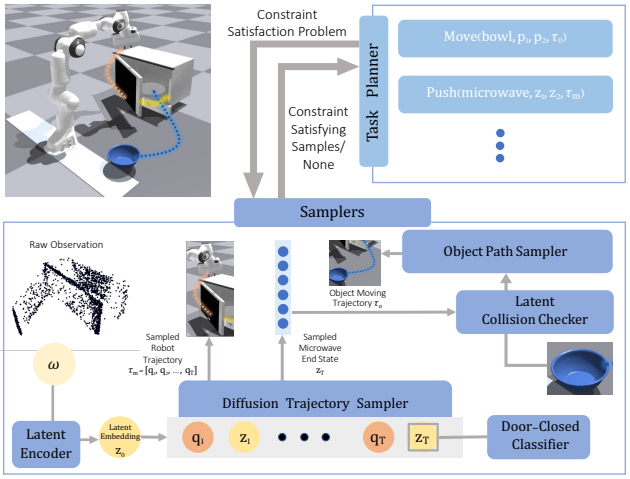


Fig. 1: DIMSAM composes diffusion models using a TAMP system towards solving multi-step manipulation problems. The planner searches for the task plan, while learned diffusion samplers find constraint-satisfying continuous values. The bottom shows a sampling procedure for finding a microwave door closing trajectory and a collision-free object stowing trajectory. A diffusion model samples a trajectory of latent microwave states $z_1, ..., z_T$ and robot configurations $q_1, ..., q_T$ that reaches a door-closed state z_T.

A TAMP solver is adept at such reasoning procedures. However, finding the continuous action parameter values is challenging. For example, to place an object into a closed microwave, the action parameters include an opened configuration of the microwave, a robot trajectory that opens the microwave to that configuration, a placement pose of the object inside the microwave, and a robot trajectory to place the object that satisfies collision constraints. The inter-dependence among the constraints makes finding satisfying values difficult. We seek to improve sampling-based TAMP by learning to generate constraint-satisfying values.

We use learned generative models to represent distributions over continuous state and action parameters. Generative models are trained on a distribution $p(X)$ of possibly complicated variables X (such as images, point clouds, or

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

trajectories), and can be queried to produce samples $x \sim p(X)$ at test time. Specifically, we use *diffusion models* [2], [3], and issue conditional sample queries during inference.

The flexibility in generative models is also valuable in cases of increased partial observability. Learned generative samplers can be conditioned directly on an image or point cloud. We apply diffusion models to point cloud observations with severe partial observability and show the generalizability through real-world verification.

We propose to use diffusion models as samplers for TAMP under partial observability (DIMSAM). DIMSAM (Fig. 1) leverages both the flexibility of generative models in action parameter sampling and the joint searching and sampling capability of TAMP systems. The system searches for feasible task plans, with continuous values suggested by learned samplers. By composing these generative models using the TAMP framework, our system can solve a variety of multi-step manipulation problems under partial observability.

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. These learned models are composed through the TAMP framework, enabling simultaneous search over constraint structure and sampling on target distribution.
2) We define these constraints on a latent embedding of object state, allowing them to be applied to previously unseen articulated objects with no known models.
3) We showcase our approach on articulated object manipulation tasks that require multi-step reasoning and verify a learned sampler in the real world without finetuning.

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], [6]. Traditionally, these models are specified by a human; however, increasingly, aspects of these models, such as constraints [7], samplers [8], [9], predicates [10], and parametric operators [11], have been learned. An earlier work [5] extended TAMP methods to manipulation without shape models, with a perception module for state estimation. Our method incorporates learned distributional information into the samplers, and also addresses articulated objects.

There is also related work on learning for other manipulation planning frameworks [12], [13], [14], but usually not considering complicated objects or partial observability. Our method uses a generative diffusion model to learn samplers based only on the partially observed state of the world, and solve TAMP problems in a hybrid space of learned latent and human-specified states.

Generative models [2], [3] have been used in decision making [15], [16] and policy learning [17], [18], [19]. Compared to non-generative trajectory modeling methods [20], [21], [22], they are usually better at capturing the multi-modalities in the training data. It has also been used to learn the distribution of parameters for individual samplers [23], [24], [25], [26], [27]. In this paper, we apply a similar model but focus on how to compose the learned models using a TAMP system. Previous work usually assumes a given task plan or a constraint graph. Our goal is to use the learned model *combinatorially* with other samplers to solve multi-step planning tasks, in a framework that *jointly searches for satisfiable task plans and samples constraint-satisfying values*.

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 can't be directly applied.

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 are constraints 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* π, 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 a 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* [28] 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 grasp pose of an object, become the inputs to another, *e.g.*, an inverse kinematics (IK) solver. A *conditional sampler* for a constraint c with m arguments is a function from α, a tuple of $k < m$ argument values, to a generator of tuples β_i. Each β is of length $m - k$. When concatenated with the values α, 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 to generate values under certain constraints, such as 1) generating stable placement and grasp poses, 2) checking collisions at state s, 3) generating desirable door states (open or closed), and 4) modeling the contact dynamics of doors. When the world is known, samplers 1-3 can be engineered. However, engineering sampler 4 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 1-3 themselves is challenging.

In our approach, we use diffusion models to learn conditional samplers that represent generative models of action constraints. 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. The diffusion sampling procedure can also be biased towards generating conditional output by having a classifier as guidance.

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. Diffusion Models

We use diffusion models as learned samplers. Diffusion models [2], [3] are a class of generative models that produce samples $x^{(0)}$ from a learned distribution $p(x)$ by iteratively applying a denoising procedure $p(x^{(t-1)}|x^{(t)})$, starting from $x^{(T)}$, a sample from the Gaussian noise distribution. 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, μ_θ and Σ_θ are time-conditional functions with learnable parameter θ. During training, a forward process gradually adds 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.

B. 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* [29] or *classifier-free* [30] 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 all potential constraint types are known when training the models, but would like to compose new constraints directly with the existing models. Thus, we opt to use *classifier-based* guidance as it allows us to combine the unconditional diffusion model with new classifiers after it is trained.

For classifier guidance, as shown by Dhariwal *et al.* [29], 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.

C. 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 ω 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 3D point is challenging and inefficient.

For a more compact encoding and to make the constraint learning problem easier, we train a point cloud encoder ϕ_{enc} to encode the observation ω into latent state z. Observed partial point cloud $\omega \in \mathcal{R}^{N\times 3}$ is represented as a latent vector $z \in \mathcal{R}^{d_z}$, where d_z is the latent dimension.

D. 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) Pushing Trajectory Sampler: Non-prehensile motions that involve continual contact such as pushing are difficult to plan for, especially if the object is not known *a priori*. We hope a pushing trajectory sampler can model the constraints and dynamics of a robot interacting with an articulated object via pushing.

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

The vanilla DiffPush sampler models a distribution of trajectories. The trajectory contains both the robot configuration q and latent state z of object o, over T time steps. Different from the commonly used diffusion policy, this is a transition model, and not a policy directly.

Recall that we can bias the diffusion sampling, drawing a conditional sample from this trajectory distribution can be done by adding such biases on some of these variables $(z_1, ..., z_T, q_1, ..., q_T)$. We can apply conditionings to different sets of variables, which creates a set of different samplers. If the start of the trajectory z_1 is known and we want to predict the states in the following time steps, we can create a *forward* sampler $p(q_1, z_2, q_2, ..., z_T, q_T | o, z_1)$ by enforcing $z_1 = \mathbf{z_1}$. Similarly, when the end state z_T is given, we can have a *backward* sampler $p(z_1, q_1, z_2, q_2, ..., q_T | o, z_T)$ that infers the *pre-image* of latent state z_T. Or, if one wants to sample the possible trajectories that can change the object state from $z_1 = \mathbf{z_1}$ to $z_T = \mathbf{z_T}$, we can supply a *bi-directional* sampler $p(q_1, z_2, q_2, ..., q_{T-1} | o, z_1, z_T)$ that fills in possible transitions.

In an H-CSP solving context, depending on the variable ordering, the DiffPush sampler can be applied in multiple ways. For example, if the CSP solver samples a target state z_T prior to sampling a pushing trajectory, where z_T needs to satisfy another constraint c_ψ, z_1 is observed, then the *bi-directional* sampler can be used to find a robot-object trajectory

$$p(q_1, z_2, q_2, ..., q_{T-1} \mid o, z_1, [c_\psi(o, z_T) = \textbf{True}]). \quad (4)$$

2) Object State Classifiers: Often, the goal conditions that define S_* require specific objects to be in a configuration that is semantically meaningful for a human, for example, a state that a human considers door being open DoorOpen[·] or closed Doorclosed[·] based on a point cloud observation. To model this, we learn classifiers $c_*(o, z)$ on the object o and the latent state z. These classifiers can be applied on already-sampled object state z, to filter invalid samples, as a rejection sampler. Alternatively, with a diffusion sampler, it can be used to bias the sampling while generating conditional samples, as described in Sec. IV-B. For example, the gradient from a DoorClosed[·] classifier can guide a DiffPush sampler to generate robot and microwave trajectory, that the door ends at a closed state, as shown in Fig. 1.

3) 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 $c_\zeta(o, z, o_2, p_2)$ for constraint PairwiseCollision 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 Π using PDDLStream [28], an extension of Planning Domain Definition Language (PDDL) [31] that supports planning with continuous values using sampling operations. Planning state variables and action constraints are represented using *predicates*, which are essentially named classifiers. They are grounded using the classifier introduced in the previous section.

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 and an object o_1 is stowed in the microwave can be defined by:

$$\exists\, z.\, \texttt{AtLatentState}(o_m, z) \wedge \underline{\texttt{DoorOpen}(o_m, z)} \wedge \texttt{Stowed}(o_1),$$

where the predicate DoorOpen is a learned classifier.

B. Actions with Learned Constraints

The push action involves the major learned constraints in this domain (Fig. 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 DiffPush. DiffPush can be of different forms depending on the variable ordering while solving the CSP, such as the aforementioned forward sampler and bi-directional sampler.

push$(o, z_1, q_1, ..., z_T, q_T)$
con: [$\underline{\texttt{DiffPush}(o, z_1, q_1, ..., z_T, q_T)}$,
$\underline{\neg\texttt{Unsafe}(o, z_1)}$, ..., $\underline{\neg\texttt{Unsafe}(o, z_T)}$]
pre: [AtLatentState(o, z_1), AtRobotConf(q_1)]
eff: [AtLatentState(o, z_T), AtRobotConf(q_T),
$\neg$AtLatentState(o, z_1), $\neg$AtRobotConf(q_1)]

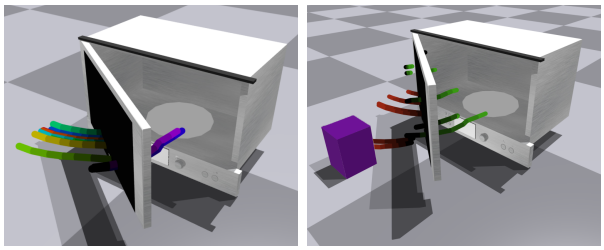
Fig. 2: The push action description.

Unsafe(o, z) imposes the collision-free constraint on the state z on all other objects, which is evaluated by the learned classifier PairwiseCollision. Pairwise collision is checked between o and all other objects in the domain.

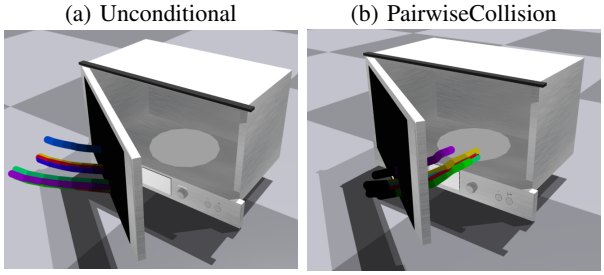
$$\texttt{Unsafe}(o, z) \equiv \exists\, o_2, p_2.\, \texttt{AtPlace}(o_2, p_2) \wedge \underline{\texttt{PairwiseCollision}(o, z, o_2, p_2)}.$$

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.

$$\pi = [\texttt{MoveArm}(\mathbf{q_0}, \tau_1, q_1), \texttt{Push}(o_m, \mathbf{z_1}, q_1, ..., z_2, q_2), \texttt{MoveArm}(q_2, \tau_2, q_3), \texttt{Pick}(\texttt{block}, g, \mathbf{p_0}, q_3), \texttt{MoveArm}(q_3, \tau_3, q_4), \texttt{Place}(\texttt{block}, g, p_*, q_4)]$$



(a) Unconditional (b) PairwiseCollision



(c) DoorOpen (d) Doorclosed

Fig. 3: Samples from the `DiffPush` model. (a) No condition except for known z_1. (b) Checking collision with the purple obstacle using classifier `PairwiseCollision`. Rejected samples are colored red. (c) (d) Classifier-guided conditional sampling with `DoorOpen` and `Doorclosed`.

C. Environment and Data Collection

We perform our experiments in the IsaacGym [32] physics simulator and verify the learned sampler and classifier in the real world. To learn the data distribution that satisfies `DiffPush` constraint, we generate a training set $\mathcal{D}$ of push action instances by simulating manipulation trajectories in IsaacGym. The manipulation trajectories are generated using a script policy that has access to the ground truth state, with known object models and contact forces. An action sequence consists of end-effector waypoints pushing perpendicular to the segmented door surface. In the training set, we use 10 microwave assets from the PartNet [33] dataset. We capture the depth images from randomized viewpoints. There are 101 valid trajectories in total, which are later randomly clipped into shorter segments for training.

During data collection, we treat the robot as a disembodied gripper, 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 to be flexibly integrated into full arm motion using inverse kinematics and motion planner in a TAMP framework. Our real-world experiments demonstrate such flexibility by composing the same learned model with robot grippers that have different geometries.

D. Training Details

To train the point cloud encoder ϕ_{enc}, we use the aforementioned trajectory dataset, which contains 4746 point clouds. We use PointNet++ [34] as the encoder network ϕ_{enc} and use the same decoder ϕ_{dec} as Cai *et al.* [35]. 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.* [35]. 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 a U-Net structure similar to that in Diffuser [15], with 250 diffusion steps, trajectory length 8, and batch size 32. Microwave state (open/close) classifiers are 3-layer MLPs. The inputs to the collision network `PairwiseCollision` are latent point cloud embedding, obstacle location, and obstacle size approximated as bounding boxes.

Method	**Mean Error (Stderr)**
Regression model	0.127 (0.021)
EBM	0.191 (0.021)
Diffusion	**0.094 (0.009)**

TABLE I: Mean error to the target angle (in radians) after executing the trajectory in simulation. Our method (Diffusion) outperforms the two baselines.

VI. Experiments

We carry out standalone evaluations on our learned samplers, and further evaluate them in the integrated TAMP setting with joint searching and sampling. The evaluations are done in IsaacGym simulator. We also apply our learned sampler on a real robot, as described in section VII.

A. Generating Samples with Specific Constraint

To verify that our learned push sampler can generate valid and accurate samples, we impose tight constraints on the initial and target state, and exploit the *bi-directional* sampler to fill in intermediate steps in trajectories.

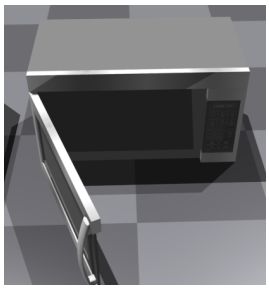
$$p(q_1, z_2, q_2, ..., q_{T-1}|o, z_1, z_T)$$

We compare with other types of non-diffusion-based models and report the error on door angle after executing the sampled robot trajectory $[q_1, \ldots, q_{T-1}]$ in simulation.

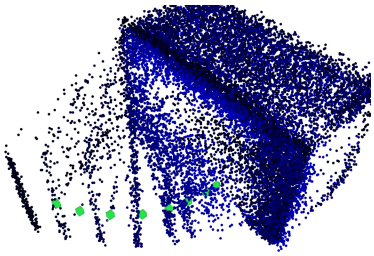
The initial and target microwave door angles are randomly sampled. We capture the point cloud at such configurations ω_1, ω_T from a random viewpoint. The point cloud observations are encoded into latent z_1, z_T and used as conditions to sample the rest of the trajectory from the diffusion model.

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).

Table I shows the mean (and standard error) of the absolute distance between the target angle and the actual ending angle on 100 testing trajectories. Results are shown in radians. The diffusion model has the lowest error. Compared to EBM, the diffusion models are easier to train and have a more stable gradient, which is also observed in [17]. Though the regression model is also stable during training, the result is deterministic given the same point cloud embeddings z_1, z_T, which is not desirable as a sampler according to our analysis in Sec. III-B. In contrast, the diffusion model can generate more diverse samples that satisfy the given constraints. See Fig. 3 for an illustration.



(a) Initial microwave state.



(b) Predicted microwave trajectory.

Fig. 4: One sampled trajectory of the microwave from the initial state (left figure) to the "fully closed" state. Sampled latent state z is decoded into a point cloud and rotated for better visualization.

B. Generating Conditional Samples with Classifier Guidance

The previous section evaluates our model on a given goal state z_T. We want to verify that our model can also generate trajectories conditioned on a more abstract goal,

$$p(o, q_1, z_2, q_2, ..., q_{T-1} \mid o, z_1, [c_\psi(o, z_T) = \textbf{True}])$$

where the target ending state is given in the form of a constraint. Any trajectory $q_1, ..., q_{T-1}$ leading to a state z_T such that $c_\psi(o, z_T) = \textbf{True}$ is a constraint-satisfying sample.

We evaluate the learned model with classifier-guided conditional sampling, where c_ψ is given in the form of named classifiers, according to Eq. 2. We train three classifiers to model the point cloud as being "fully closed", "fully open", "half closed", and "half open". The corresponding semantics are door angle less than 0.2, greater than 1.4, less than 1.2, and greater than 1.2 radians. We evaluate 100 random initial configurations of the microwave and report the success rate of the ending state after executing the sampled trajectory $q_1, ..., q_{T-1}$ (Table II). Qualitative results are shown in Fig. 3, with the door being fully closed and fully opened. Fig. 4 shows the entire sampled trajectory with classifier guidance of "fully closed". The green boxes show the action waypoints $q_1, ..., q_{T-1}$.

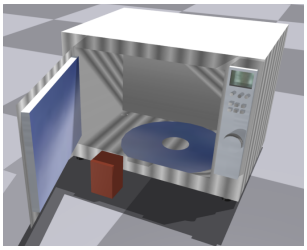
Goal	Fully Closed	Fully Open	Half Closed	Half Open
Success	94%	92%	100%	90%

TABLE II: Success rates for achieving abstract goals using classifier guidance.

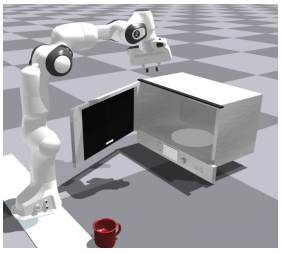
Our model achieves a high success rate on all semantic goals. Further, the classifiers are trained completely independent of the diffusion trajectory sampler. This is valuable for generalization to new domains with novel constraint types. One can re-use the unconditional trajectory model and only learn the classifier that encodes the semantics of the new constraint.

C. TAMP: Joint searching and sampling

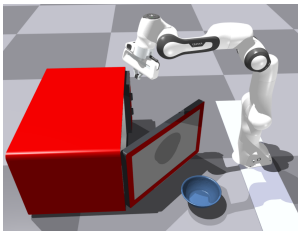
In prior experiments, we verify the sampling capability of our learned models. In the following section, we test the learned models in a complete TAMP system. Diffusion models and learned classifiers are combined with other samplers such as IK and motion planners, in order to solve a multi-step manipulation task. The end-effector in previous experiments is simplified as a moving gripper. In this experiment, we add the full-arm motion and enforce additional constraints such as the sampled end-effector poses being reachable. These additional constraints need to be considered jointly. If none of the sampled trajectories satisfy all constraints, we either draw more samples, or search for a new task plan that has a different constraint relation. We use PDDLStream to jointly search the task plan and make calls to corresponding samplers.



(a) The *Close* task.



(b) The *Stow-Close* task.



(c) The *Stow-Close-B* task.

Fig. 5: The initial states in the (a) *Close*, (b) *Stow-Close*, (c) *Stow-Close-B* simulated tasks.

We evaluate three tasks, shown in Fig. 5. 1) *Close*: The goal state is the microwave door at the closed location. The initial state is the door at an opened position, where there is an obstacle blocking the door from directly being closed. 2) *Stow-Close*: The goal is to have an object stowed in the microwave and the door is closed at the end. The object is initialized to be at a fixed location that won't block the microwave. 3) *Stow-Close-B*: 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. As in the previous experiments, the planner only receives a partial point cloud of the microwave captured from a random viewpoint. The initial location and geometry of the object to be stowed and the obstacle are known. We use a Franka Emika robot arm with a mobile base link along the X-axis. The predicted waypoints $q_1, ..., q_{T-1}$ are set as the target endpoint of the gripper. We use the robot URDF to compute IK solutions for the arm. For collision checking between the robot arm and a predicted microwave state z, we approximate links of the arm using bounding boxes and query the learned collision checker. We use the learned latent collision checker for all predicted states and the partial point cloud for the initial state. We compute the motion plans using a bidirectional RRT.

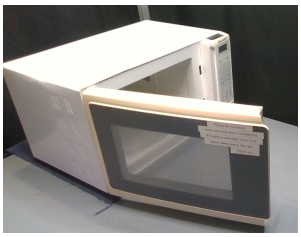
Task	#Actions	PSR	ESR
Close	1.93	1.00	0.60
Close^{c*}	1.93	1.00	0.87
Stow-Close	1.93	0.93	0.71
Stow-Close^{c*}	2.00	1.00	1.00
Stow-Close-B^{c*}	4.00	0.87	1.00

TABLE III: Results on joint planning using PDDLStream. Task names with the superscript $\cdot^{c*}$ use conditional samplers. PSR stands for planning success rates. ESR stands for execution success rates.

We conduct an ablation on rejection sampling and classifier-guided conditional sampling. Rejection sampling refers to the variant that uses a *forward* sampler $p(q_1, z_2, q_2, ..., z_T, q_T | o, z_1)$ and testing post-hoc whether $c_\psi(o, z_T) =$ **True**. Conditional sampling refers to using the goal-conditioned sampler as used in Sec. VI-B

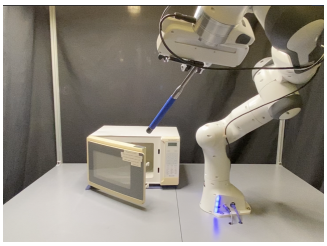
We use the *Adaptive* algorithm in PDDLStream with a *search-sample* ratio of 1/15 for all tasks. We use weighted A* and fast-forward heuristic. Planning success rates (PSR) and execution success rates (ESR) are shown in Table III. PSR refers to the success rate during planning - the ratio of problems solved by the planner (which does not necessarily lead to successful execution). ESR refers to that when executing a trajectory, the resulting state actually achieves the goal, among the ones solved by the planner. We evaluate each task for 15 runs. Average high-level action numbers are reported based on solved runs. The high-level actions the planner considers include `Move(·)`, `Stow(·)`, `Push(·)`. Even without considering which object to apply the action, there are 3 possible combinations for plan length 1 and 3^4 combinations for a plan with 4 actions. Examples of the final plans are available at the website.

Behavior Analysis In the first task, the first plan found by the planner will try to close the door directly `[Push(`o_m`)]`. Constraints in this plan include [`DoorClosed(`o_m, z_T`)`, `DiffPush(`$o_m, z_1, q_1, \ldots, z_T$`)`, $\neg$`Unsafe(`o_m, z_*`)`], etc. These constraints are not satisfiable, since there is an obstacle blocking the door from closing, so there is no collision-free path for the microwave to close, without moving the blocking object first ($\neg$`Unsafe(`o_m, z_*`)` is unsatisfiable). With returned failures from the samplers, the planner will start searching for an alternative plan. The most typical satisfiable plan found is `[Move(`o_1`), Push(`o_m`)]`. We found the classifier-guided sampling Close^{c*} to have a higher execution success rate, as guided sampling of the diffusion model (as in Eq. 4) tends to generate higher quality samples than the *forward* sampler. In *Stow-Close*, the planner needs to reason about the order to achieve the subgoals in `[Push(`o_m`), Stow(`o_1`)]`. Closing the door prior to stowing object can achieve one subgoal, but making the sampling of another subgoal unsatisfiable, without undoing the first action. The TAMP planner can reason about such constraints and still has a high PSR. We also have the same observation here that the ESR of using conditional sampler is higher than that of the rejection sampling. The third task is the most challenging one. A precondition of the 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. Two out of 15 tests does not return a valid plan.

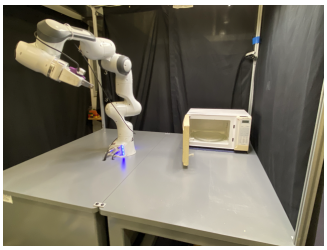


(a) Observed RGB image. Models only use the depth data.

(b) Observed point cloud. Rotated to top-down for better visualization.



(c) The close door task.



(d) The open door task.

Fig. 6: Real world deployment setup and model observation.

Among the execution failure modes, we have observed that the sampled trajectory may be too close to the joint axis of the door and making it physically infeasible. We will consider incorporating uncertainty or effort estimates from learned samplers in future work.

It is important to note that this is a problem that cannot be addressed by classical TAMP methods, because the kinematic and shape models of the microwave are unknown. In addition, note that no new learning was required to solve 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 work poorly on multi-step problems that require geometric and constraint reasoning 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 D435i depth camera mounted on the Franka gripper. The point cloud of the microwave is segmented by removing the table plane. We apply the learned `DiffPush` model trained in simulation on the observed point cloud directly without any finetuning, demonstrating zero-shot sim-to-real transfer. In the door closing and opening tasks, we use conditional sampling based on `Doorclosed` and `DoorOpen` classifiers trained in simulation. The planner searches for trajectories that satisfy reachability and collision-free constraints.

In the door-opening task, the Franka robot gripper is too wide to fit in between the microwave door and the microwave base. We augment the robot with a tool that it can use to

make contact with the microwave. The waypoints predicted by our diffusion models are set as the target locations of the tool endpoint. Since the action representation in the learned model is simplified as waypoints, we don't need to re-train the model but only need to modify the IK component.

The observed RGB image and point cloud are shown in Fig. 6. The RGB image itself is for illustration only and is not used by the model. Due to the reflective material of the microwave and the glass used in the microwave door, there are a lot of missing depth readings, as shown in Fig. 6b. Despite this severe partial observability, our model is still able to generate actionable target samples. Videos showing the robot solving these tasks are available at: https://sites.google.com/view/dimsam-tamp. Thanks to the modular design of the TAMP system and the choice of observation and action space in learned models, we are able to deploy the system in the real world without finetuning.

VIII. Conclusion

We use diffusion models for sampler learning and compose the learned samplers in a TAMP framework. Importantly, our framework can *both* reason about the set of constraints that need to be satisfied, and use the learned models in drawing constraint-satisfying samples. We have showed that conditional sampling on learned models is efficient and effective. We instantiated an example of such samplers in an articulation manipulation domain through learning pushing constraints. Finally, we compared our diffusion approach with several baselines in simulation, demonstrated it within a TAMP system for completing multi-stage tasks, and ultimately deployed it in the real-world. We believe this strategy can be applied in broader domains.

IX. Acknowledgement

We gratefully acknowledge support from NSF grant 2214177; from AFOSR grant FA9550-22-1-0249; from ONR MURI grant N00014-22-1-2740; from ARO grant W911NF-23-1-0034; from the MIT Quest for Intelligence; and from the Boston Dynamics Artificial Intelligence Institute.

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*, 2021.

[2] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *ICML*, 2015.

[3] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *NeurIPS*, 2020.

[4] M. Toussaint, "Logic-geometric programming: an optimization-based approach to combined task and motion planning," in *IJCAI*, 2015.

[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 *ICRA*, 2022.

[6] 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*, 2020.

[7] J. Mao, T. Lozano-Perez, J. B. Tenenbaum, and L. P. Kaelbing, "PDSketch: Integrated Domain Programming, Learning, and Planning," in *NeurIPS*, 2022.

[8] B. Kim, L. P. Kaelbling, and T. Lozano-Pérez, "Guiding search in continuous state-action spaces by learning an action sampler from off-target search experience," in *AAAI*, 2018.

[9] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Learning compositional models of robot skills for task and motion planning," *IJRR*, 2021.

[10] T. Migimatsu, W. Lian, J. Bohg, and S. Schaal, "Symbolic state estimation with predicates for contact-rich manipulation tasks," in *ICRA*, 2022.

[11] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling, "Learning neuro-symbolic skills for bilevel planning," in *CoRL*, 2022.

[12] A. H. Qureshi, A. Mousavian, C. Paxton, M. C. Yip, and D. Fox, "NeRP: Neural rearrangement planning for unknown objects," in *RSS*, 2021.

[13] D. Driess, J.-S. Ha, M. Toussaint, and R. Tedrake, "Learning models as functionals of signed-distance fields for manipulation planning," in *CoRL*, 2022.

[14] 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 *ICRA*, 2021.

[15] M. Janner, Y. Du, J. Tenenbaum, and S. Levine, "Planning with diffusion for flexible behavior synthesis," in *ICML*, 2022.

[16] A. Ajay, Y. Du, A. Gupta, J. Tenenbaum, T. Jaakkola, and P. Agrawal, "Is conditional generative modeling all you need for decision-making?" in *ICLR*, 2023.

[17] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song, "Diffusion policy: Visuomotor policy learning via action diffusion," in *RSS*, 2023.

[18] U. A. Mishra, S. Xue, Y. Chen, and D. Xu, "Generative skill chaining: Long-horizon skill planning with diffusion models," in *CoRL*, 2023.

[19] J. Carvalho, A. T. Le, M. Baierl, D. Koert, and J. Peters, "Motion planning diffusion: Learning and planning of robot motions with diffusion models," in *IROS*, 2023.

[20] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," in *NeurIPS*, 2021.

[21] 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," 2021.

[22] F. Liu, H. Liu, A. Grover, and P. Abbeel, "Masked autoencoding for scalable and generalizable decision making," in *NeurIPS*, 2022.

[23] Z. Yang, J. Mao, Y. Du, J. Wu, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling, "Compositional Diffusion-Based Continuous Constraint Solvers," in *CoRL*, 2023.

[24] J. Mendez-Mendez, L. P. Kaelbling, and T. Lozano-Pérez, "Embodied lifelong learning for task and motion planning," in *CoRL*, 2023.

[25] W. Liu, Y. Du, T. Hermans, S. Chernova, and C. Paxton, "Structdiffusion: Language-guided creation of physically-valid structures using unseen objects," in *RSS*, 2023.

[26] A. Simeonov, A. Goyal, L. Manuelli, L. Yen-Chen, A. Sarmiento, A. Rodriguez, P. Agrawal, and D. Fox, "Shelving, stacking, hanging: Relational pose diffusion for multi-modal rearrangement," in *CoRL*, 2023.

[27] J. Urain, N. Funk, J. Peters, and G. Chalvatzaki, "Se(3)-diffusionfields: Learning smooth cost functions for joint grasp and motion optimization through diffusion," in *ICRA*, 2023.

[28] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "PDDLStream: Integrating Symbolic Planners and Blackbox Samplers," in *ICAPS*, 2020.

[29] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," in *NeurIPS*, 2021.

[30] J. Ho and T. Salimans, "Classifier-free diffusion guidance," in *arXiv preprint arXiv:2207.12598*, 2022.

[31] M. Fox and D. Long, "Pddl2.1: An extension to pddl for expressing temporal planning domains," *JAIR*, 2003.

[32] 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," in *arXiv preprint arXiv:2108.10470*, 2021.

[33] 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 *CVPR*, 2019.

[34] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *NeurIPS*, 2017.

[35] R. Cai, G. Yang, H. Averbuch-Elor, Z. Hao, S. Belongie, N. Snavely, and B. Hariharan, "Learning gradient fields for shape generation," in *ECCV*, 2020.