

# PRESTO: Fast motion planning using diffusion models based on key-configuration environment representation

Mingyo Seo<sup>1\*</sup>, Yoonyoung Cho<sup>2\*</sup>, Yoonchang Sung<sup>1</sup>, Peter Stone<sup>1,3</sup>, Yuke Zhu<sup>1†</sup>, and Beomjoon Kim<sup>2†</sup>

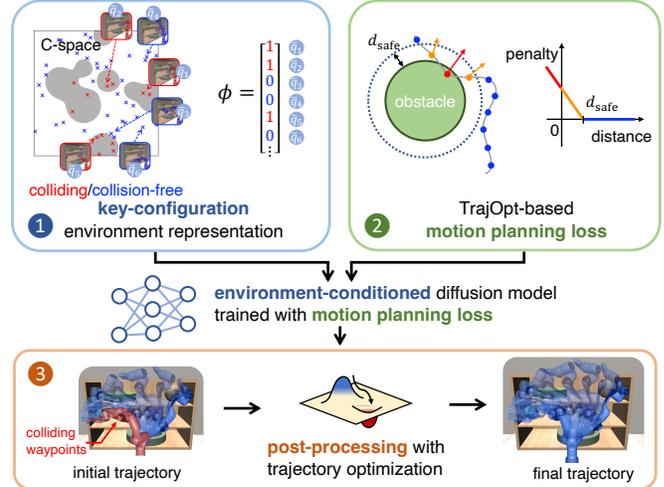
**Abstract**—We introduce a learning-guided motion planning framework that provides initial seed trajectories using a diffusion model for trajectory optimization. Given a workspace, our method approximates the configuration space (C-space) obstacles through a key-configuration representation that consists of a sparse set of task-related key configurations, and uses this as an input to the diffusion model. The diffusion model integrates regularization terms that encourage collision avoidance and smooth trajectories during training, and trajectory optimization refines the generated seed trajectories to further correct any colliding segments. Our experimental results demonstrate that using high-quality trajectory priors, learned through our C-space-grounded diffusion model, enables efficient generation of collision-free trajectories in narrow-passage environments, outperforming prior learning- and planning-based baselines. Videos and additional materials can be found on the project page: <https://kiwi-sherbet.github.io/PRESTO>.

## I. INTRODUCTION

Motion planning involves finding a smooth and collision-free path in a high-dimensional configuration space (C-space). Classical motion planning algorithms typically use either sampling-based methods [16, 22, 23] or optimization-based methods [29, 33] to address motion planning across various domains. However, in high-dimensional C-spaces with narrow passages, sampling-based methods incur high computational costs due to large search-space and small volume of solutions. While optimization-based methods can be an alternative, such methods are sensitive to initialization and may become stuck in local optima, often failing to find a feasible path. Consequently, both approaches have limitations when dealing with complex motion planning problems under restricted computational resources.

Recent works leverage generative models to directly learn trajectory distributions instead [4, 10, 14]. By casting motion planning as sampling from a learned distribution, these models can efficiently generate trajectories within a consistent compute budget. However, they often struggle to generalize to new, complex C-spaces, resulting in high collision rates in the generated trajectories, because most of these approaches use workspace as an input to neural networks, instead of C-space.

Instead, we propose to represent the environment in terms of key configurations [19], which are a sparse set of task-related configurations from prior motion planning data. The resulting model no longer needs to learn a generalizable



**Fig. 1: Overview of PRESTO.** PRESTO aims to generate collision-free trajectories in complex, unseen C-spaces. First, we approximate these C-spaces using key configurations from prior data and generate trajectories based on this representation. Conditional diffusion models, trained with motion planning loss, provide initial solutions, which are then refined through trajectory optimization.

mapping between workspace and C-space obstacle representations, leading to improved generalization and reduced training complexity.

Another challenge is designing a training objective for diffusion models tailored to motion planning. Existing diffusion models for motion-planning use DDPM-based loss [4, 9, 14], which focuses on reconstruction quality [31]. However, the reconstruction objective does not account for underlying task constraints, resulting in degraded performance for tasks requiring precise outputs and complex constraints [8]. To overcome this, we incorporate TrajOpt-inspired motion-planning costs [33] directly into the training pipeline of diffusion models. By training the model to directly minimize trajectory-optimization costs associated with collision avoidance and trajectory smoothness, the model learns to generate smooth and collision-free trajectories. Further, to ensure that the trajectory satisfies hard constraints such as collision avoidance, we feed the output of the diffusion model to trajectory optimization as an initial solution.

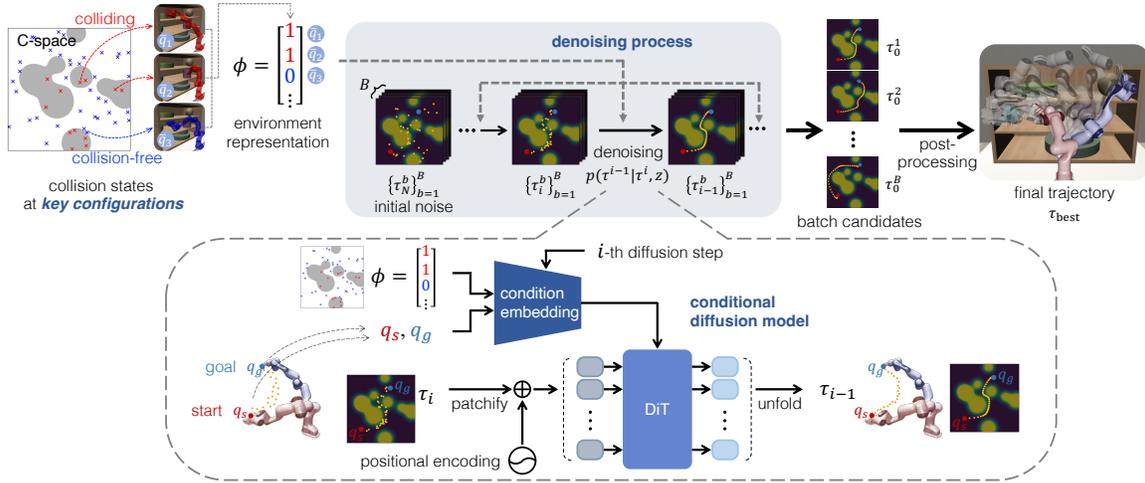
We call our combined framework PRESTO (Planning with Environment Representation, Sampling, and Trajectory Optimization). Figure 1 summarizes our framework: 1) an environment representation based on key configurations (blue block), 2) a training pipeline for diffusion models that directly integrates trajectory optimization costs (green block), and 3) a diffusion-based sampling-and-optimization framework for motion planning, where the diffusion model

<sup>1</sup>The University of Texas at Austin,

<sup>2</sup>Korea Advanced Institute of Science and Technology, <sup>3</sup>Sony AI,

\*Equal contribution, †Equal advising.

Correspondance: beomjoon.kim@kaist.ac.kr



**Fig. 2: Trajectory generation pipeline of PRESTO.** We obtain the environment representation for an unseen problem by checking the collision states at the key configurations used during training. Using the trained conditional diffusion model, we generate multiple trajectories conditioned on this representation and then select the least-colliding trajectory after post-processing.

provides initial trajectories for trajectory optimization (orange block). We evaluate PRESTO in simulated environments where a robot operates in a fixed scene populated with randomly shaped and arranged objects. The results show that the synergy between our diffusion model’s high-quality trajectory priors and the trajectory optimization post-processing efficiently generates collision-free trajectories in narrow passages within a limited compute budget.

## II. RELATED WORK

Several approaches use learning to guide motion planning, where they complement motion planners with learned models for collision-checking [5, 6, 11, 17, 24, 41] or sampling promising configurations [2, 12, 13, 18, 20, 21, 36, 38].

One line of work learns collision checkers to expedite motion planning. While one body of work progressively learns collision models in a given environment as Gaussian mixtures [11] or kernel perceptrons [6, 41], others learn deep neural network models from large offline datasets. Kew et al. [17] trained a neural network to estimate the clearance of robot configurations in a fixed environment; Danielczuk et al. [5] generalize collision predictions to diverse environments by training a neural network to estimate collision states from object and scene point clouds. Recently, Murali et al. [24] extended neural collision detectors to a partially observable setting. Our approach can integrate with these methods for collision-checking to further enhance planning speed.

Another line of work learns to sample promising configurations, either from explicit distributions constructed via kernel density estimation [2, 13] or by using neural networks, where sequence-based models such as LSTMs [21] or a rejection-sampling policy [38] are used to learn the distribution of collision-free configurations based on the history of collision states at previously sampled configurations. Alternatively, recent works propose to use generative models to govern the sampling process. Qureshi et al. [28] maps point-cloud inputs to the next sampling configuration, stochastically applying dropout in the interim layers to generate

diverse samples. Yu et al. [36] use Graph Neural Networks to identify promising regions of a roadmap. Ichter et al. [12] and Kumar et al. [20] employ Conditional Variational Autoencoders (CVAEs) to sample task-relevant configurations from latent spaces. While these methods generate individual or sets of joint configurations, recent works propose to use diffusion models to learn the sampling distributions of trajectories to accelerate motion planning.

Diffuser [14] first explored diffusion models for generating trajectories across various start and goal configurations in a fixed environment. Subsequent work aims to generalize to unseen environments using test-time guidance [4, 30], but struggles with large environmental variations due to significant mismatches in learned trajectory distributions. Recent studies [1] address this issue by applying conditioning diffusion models on the environments. Notably, Huang et al. [10] condition on point-cloud representations to sample motion plans across diverse environments. However, these models still face challenges in generalizing to new C-spaces, as motion planning occurs in C-space, but they use workspace as input to the neural networks.

Prior work explores alternative representations for motion planning, focusing on C-space grounded approaches [15, 19]. These methods approximate complex C-space from past problems to find collision-free trajectories in unseen environments. In particular, our work incorporates a key-configuration representation inspired by Kim et al. [19], which utilizes collision states at task-related key configurations to enhance model generalization across varying C-spaces.

## III. PROBLEM DESCRIPTION

Let  $\mathcal{C}$  be a  $d$ -dimensional C-space, which is divided into two subspaces:  $\mathcal{C}_o$  representing C-space obstacles, and  $\mathcal{C}_f = \mathcal{C} \setminus \mathcal{C}_o$  representing the collision-free C-space. We denote the robot’s configuration as a  $d$ -dimensional vector  $q \in \mathcal{C}$ . A trajectory is represented as a sequence of waypoint configurations  $\tau = (q_0, q_1, \dots, q_T)$ . Given the start configuration  $q_s$

and goal configuration  $q_g$ , the objective of motion planning is to find a collision-free path  $\tau \in \mathcal{C}_f$  from  $q_s$  to  $q_g$ .

In this work, we assume we are provided with a dataset  $\mathcal{D} = \{\mathcal{D}_m\}_{m=1}^M$  that comes from solving  $M$  number of past planning problems, where each data-point  $\mathcal{D}_m = \{q_s, q_g, \tau, \mathcal{G}\}$  consists of a start configuration  $q_s$ , a goal configuration  $q_g$ , a trajectory  $\tau$ , and an environment geometry  $\mathcal{G}$ . We assume consistent environment fixtures but varying object shapes and locations across problems. We use an optimization-based planner to compute ground-truth trajectories for training data. Our goal is to develop a generative model that provides a good initial solution for trajectory optimization, even in environments with unseen obstacles and their arrangements.

#### IV. METHOD

PRESTO comprises of three key components, illustrated in Figure 1. First, we generate a set of key configurations and their collision states from the motion-planning dataset. Based on the resulting representation, we train a conditional diffusion model, incorporating trajectory optimization costs to guide the model toward smooth and collision-free trajectories. Finally, we feed the trajectory generated by the diffusion model to trajectory optimization. Each component of our framework is detailed in the following sections.

##### A. Environment Representation

We represent the environment as an approximation of its C-space using a collection of key configurations selected from the motion planning dataset. We denote the set of the key configurations as  $\{\bar{q}^k\}_{k=1}^K$ , where the number of key configurations  $K$  determines the resolution of the approximation for the environment’s C-space<sup>1</sup>. For each motion planning problem, we compute the environment representation  $\phi \in \{0, 1\}^K$  as a binary vector that specifies the collision status of key configurations  $\{\bar{q}^k\}_{k=1}^K$ .

Algorithm 1 describes our procedure for generating key configurations, which is a modified version of the original algorithm proposed in Kim et al. [18]. It takes the dataset  $\mathcal{D}$  and hyperparameters  $d_q^{\min}, d_x^{\min}, c, K$ . Here,  $d_q^{\min}$  is the minimum C-space distance between key configurations,  $d_x^{\min}$  is the minimum workspace distance for end-effector tips,  $c$  is the bound on the proportion of environments where the key configuration is occupied, and  $K$  is the number of key configurations to sample.

We initialize the key configuration set as an empty set (line 1), then sample key configurations into the buffer until the target number is reached (lines 2-11). At each step, we uniformly sample a configuration from  $\mathcal{D}$  (lines 3-4) and filter it based on three conditions (lines 5-10). To avoid duplicates, we ensure that the distance between the new key configuration and any existing ones exceeds a predefined threshold in both C-space and workspace distances (lines 5-6). Additionally, we check that the proportion of collision states across different environments falls within a given

<sup>1</sup>Larger  $K$  increases the resolution of the C-space approximation, but it also raises the computational overhead at query time.

range to prioritize informative configurations<sup>2</sup> (line 7). If the configuration meets all criteria, we add it to the key configuration buffer (lines 8-10). This process generates key configurations that effectively capture task-relevant regions of the C-space for motion planning.

##### B. Training Conditional Diffusion Models

a) *Model Architecture*: Figure 2 (bottom) illustrates our model architecture, based on the Diffusion Transformer (DiT) by Peebles et al. [27]. Our model takes as inputs the denoised trajectory at the  $i$ -th step  $\tau_i$ , the environment representation  $\phi$ , the start and goal joint configurations  $q_s$  and  $q_g$ , and the current diffusion step  $i$ . We utilize the  $v$ -prediction model [32], which has been empirically shown to improve sample quality [31].

To process the inputs, the trajectory  $\tau_i$  is first patched and projected by an MLP to create input tokens, as in DiT [27]. The diffusion step  $i$  and the start and goal configurations  $q_s$  and  $q_g$  are encoded with high-dimensional frequency embeddings [27] to capture small changes in these variables. The trajectory embeddings are used as input tokens for the transformer, while  $i, \phi, q_s$ , and  $q_g$  are incorporated as conditioning inputs to align sampled trajectories with the current scene and endpoint constraints.

The DiT features six transformer blocks for processing trajectories. Unlike conventional transformers, each transformer block in DiT incorporates an additional Adaptive Layer Normalization (AdaLN) layer, as described by Peebles et al. [27]. This layer applies additional gating, scaling, and bias transformations to the features of the output tokens, with the parameters predicted by a separate MLP based on the conditioning variables  $\gamma, g, b = \text{MLP}(i, \phi, q_s, q_g)$ . The output of each block  $x$  is then transformed as  $\hat{x} = x + \gamma \odot ((1 + g) \odot \text{LN}(x) + b)$ , where  $\text{LN}(x)$  denotes layer normalization and  $\odot$  denotes element-wise multiplication. This process enables the model to adjust its output according to the current environment and the diffusion iteration.

b) *Training with Motion-Planning Costs*: Our training objective includes three terms: *Diffusion Loss*, *Collision*

<sup>2</sup>By limiting the proportion of collision states, we filter out configurations that never result in collisions, as they provide no meaningful information about the environment.

---

#### Algorithm 1 Key-Configuration Selection.

---

**Require:** C-space/Workspace separation distance  $d_q^{\min}, d_x^{\min}$ ,  
Collision proportion bound  $c$ , Motion plan dataset  $\mathcal{D}$ ,  
Number of key configurations  $K$   
**Ensure:** Key configurations  $\{\bar{q}^k\}_{k=1}^K$

```

// Initialization
1:  $\{\bar{q}\} \leftarrow \emptyset$  ▷ Initialize key configuration buffer
// Sample and select key configurations
2: while  $|\{\bar{q}\}| < K$  do
3:    $\{\tau, q_s, q_g, \mathcal{G}\} \sim \mathcal{D}$  ▷ Sampling a motion plan instance
4:    $q \sim \tau$  ▷ Sampling a configuration
5:    $d_q = \text{MinCSpaceDistance}(\{\bar{q}\} \cup \{q\})$ 
6:    $d_x = \text{MinWorkspaceDistance}(\{\bar{q}\} \cup \{q\})$ 
7:    $p_c = \frac{1}{M} \sum_{m=1}^M \text{EnvCollision}(q, m)$ 
8:   if  $d_q > d_q^{\min}$  and  $d_x > d_x^{\min}$  and  $p_c \in (c, 1 - c)$  then
9:      $\{\bar{q}\} \leftarrow \{\bar{q}\} \cup \{q\}$ 
10:   end if
11: end while
12: return  $\{\bar{q}\}$ 

```

---

*Loss*, and *Smoothing Loss*. While *Diffusion Loss* is identical to the standard reconstruction objective in diffusion models, we add *Collision Loss* and *Smoothing Loss* to encourage the model to learn the motion planning constraints.

- **Diffusion Loss:** This term  $\mathcal{L}_{\text{diffusion}}$  represents the standard loss function used for training conditional diffusion models based on the DDPM framework.

- **Collision Loss:** Inspired by TrajOpt [33], this term encourages the model to generate trajectories that maintain a safe distance from objects and other links. It is defined as

$$\mathcal{L}_{\text{coll}} = \sum_{i,j} |d_{\text{safe}} - \text{sd}(\mathcal{A}_i, \mathcal{O}_j)|^+ + \sum_{i \neq j} |d_{\text{safe}} - \text{sd}(\mathcal{A}_i, \mathcal{A}_j)|^+,$$

where  $|\cdot|^+ = \max(\cdot, 0)$  and  $\text{sd}(\cdot) = \text{dist}(\cdot) - \text{penetration}(\cdot)$ .

Here,  $d_{\text{safe}} = 0.01$  m is the safety margin for the hinge loss. This loss is summed over each robot link  $\mathcal{A}_i$  and object  $\mathcal{O}_j$  based on the swept volume of the trajectory. The first term represents robot-environment collision between the  $i$ -th link  $\mathcal{A}_i$  and the  $j$ -th obstacle  $\mathcal{O}_j$ , while the second term denotes self-collision between different robot links  $\mathcal{A}_i$  and  $\mathcal{A}_j$ , where  $i \neq j$ .

- **Smoothing Loss:** This term penalizes the L2-norm between adjacent configurations, defined as  $\mathcal{L}_{\text{smooth}} = \sum_t |q_t - q_{t-1}|^2$ . It regulates the distances between consecutive configurations to encourage shorter and smoother trajectories for the robot.

We use a weighted sum of the loss terms to train our model:  $\mathcal{L} = w_1 \mathcal{L}_{\text{diffusion}} + w_2 \mathcal{L}_{\text{coll}} + w_3 \mathcal{L}_{\text{smooth}}$ , with  $w_1 = 1.0$ ,  $w_2 = 0.05$ , and  $w_3 = 0.005$ . The model is not highly sensitive to these values, but  $w_2$  should remain small for training stability.

### C. Trajectory Generation

Figure 2 (top) provides an overview of our trajectory generation process. The inputs  $q_s, q_g, \phi$  specify the motion-planning problem, where  $q_s$  and  $q_g$  are the start and goal configurations, and  $\phi$  is the environment representation from key configurations’ collision states. During denoising, we use batch sampling to leverage GPU parallelization, enhancing the likelihood of finding collision-free trajectories among stochastic outputs of the diffusion model. Our sampling follows the Denoising Diffusion Implicit Model [34], which accelerates the process by using fewer denoising iterations during inference compared to training. We then apply trajectory optimization to post-process the sampled trajectories and select the trajectory with the lowest collision cost.

We detail our trajectory generation in Algorithm 2. First, we compute the environment representation  $\phi$  by checking the collision status of the key configurations  $\bar{q}$  (line 1). Next, we initialize the denoising process with a batch of random noise  $\tau_N$  from an isotropic Gaussian distribution (line 2). We refine the trajectories over  $N$  iterations, predicting a denoised trajectory  $\tau_{i-1}$  from  $\tau_i$ , conditioned on  $\phi$  and the current iteration  $i$  (line 5). Endpoint constraints are applied at each step to ensure connectivity between the start  $q_s$  and the goal  $q_g$ , following the approach in Diffuser [14] (line 6). This process continues until the initial trajectory  $\tau_{\text{seed}}$  is obtained (line 9) and is used for post-processing.

In the post-processing phase, we refine the sampled trajectories using a fixed number of trajectory optimization iterations [33] to address potential collisions (line 10). We employ GPU-parallelized trajectory optimization [35] in batches, as it removes the need for data exchange between devices. Finally, we select the trajectory with the fewest colliding waypoints (line 11).

## V. EXPERIMENTS

### A. Experimental Setup

We evaluate our method on a motion planning task using the Franka Emika *Panda* robot arm [7], traversing a 3-tier shelf with various objects in simulation (Figure 3, top).

a) *Training Setup:* Our training domain consists of 5,000 environments with random placements of 1-6 objects (cuboid, cylinder, sphere), sampled uniformly within each shelf slot. We sample multiple initial and target joint positions within each environment’s workspace and plan trajectories using cuRobo [35], resulting in 50,000 environment-trajectory pairs annotated with key configuration labels as in Algorithm 1.

b) *Evaluation Setup:* The evaluation domain is generated using a procedure similar to the training domain, but with the dataset partitioned into different difficulty levels. We create a set of problems categorized into four levels, with each level comprising 180 different problems, to assess the performance of PRESTO and the baselines within scenes of varying complexity.

- **Level 1:** The shelf is empty, as shown in Figure 3 (top left). Although the environment remains consistent, the task is challenging due to the shelf’s non-convex workspace and the need to connect random start and goal configurations.
- **Level 2-3:** Each slot contains 1 object for *Level 2* and 2 objects for *Level 3*, adding complexity to the C-space and requiring environment-conditional collision-free trajectory generation, as shown in Figure 3 (top center).
- **Level 4:** Each slot contains 3-4 objects, as shown in Figure 3 (top right). These environments are the most challenging due to narrow passages between obstacles, increased C-space complexity, and slower collision-checking and distance calculations among many objects.

---

### Algorithm 2 PRESTO Trajectory Generation.

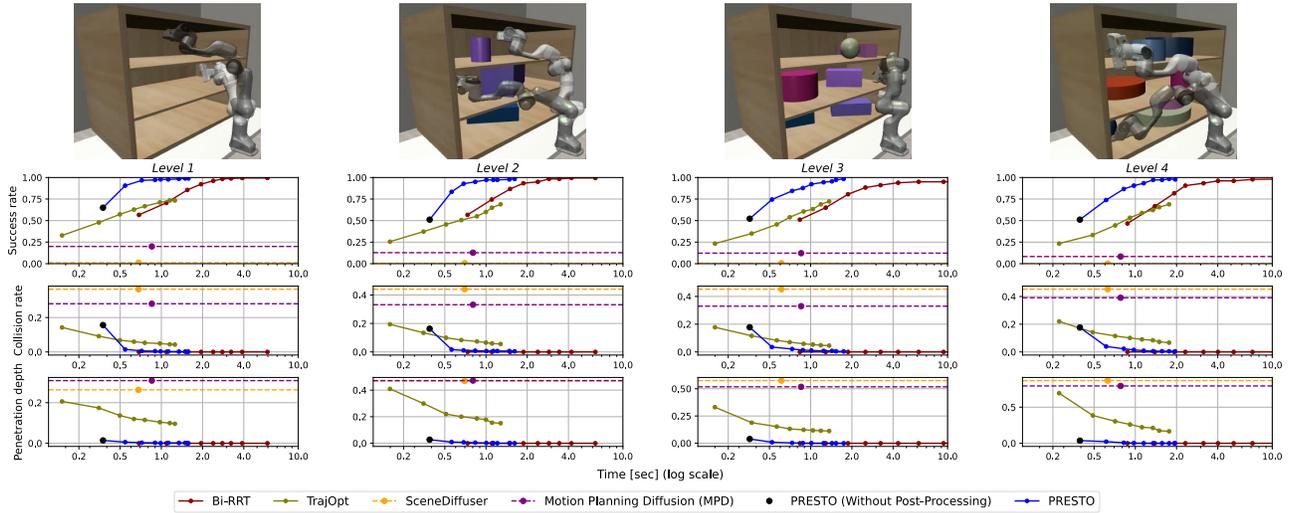
---

```

Require: Start/Goal configurations  $\{q_s, q_g\}$ , Environment  $\mathcal{G}$ ,
Key configurations  $\{\bar{q}^k\}_{k=1}^K$ , Diffusion model  $\mu_\theta(\cdot)$ ,
Noise schedule  $\{\sigma_i\}_{i=1}^N$ 
Ensure: Output trajectory  $\tau_{\text{best}}$ 
// Compute environment representation
1:  $\phi = \text{CheckCollision}(\mathcal{G}, \{\bar{q}^k\}_{k=1}^K)$ 
// Batched denoising with reverse process
2:  $\{\tau_N^b \sim \mathcal{N}(\mathbf{0}, \mathbf{I})\}_{b=1}^B$   $\triangleright$  Sample batch of initial trajectories
3: parallel for  $b = 1, \dots, B$  do
4:   for  $i = N, \dots, 1$  do
5:      $\tau_{i-1}^b \sim \mathcal{N}(\mu_\theta(\tau_i^b, \phi, i), \sigma_i)$ 
6:      $q_0 \leftarrow q_s, q_T \leftarrow q_g$   $\triangleright$  Apply endpoint constraints  $q_s, q_g$ 
7:   end for
8: end parallel
9:  $\tau_{\text{seed}} = \{\tau_0^b\}_{b=1}^B$   $\triangleright$  Batch-sized sampled trajectories
// Post-processing
10:  $\tau_{\text{opt}} = \text{TrajectoryOptimization}(\tau_{\text{seed}})$ 
11:  $\tau_{\text{best}} = \text{BestTrajectory}(\tau_{\text{opt}})$ 
12: return  $\tau_{\text{best}}$ 

```

---



**Fig. 3: Main results.** We report the success rate (%), collision rate (%), and penetration depth (m) across 180 problems. **(Top)** The evaluation environments feature consistent 3-tier shelf fixtures, with randomized object positions that vary across levels. **(Bottom)** We show PRESTO’s performance changes across domains and computational budgets compared to the baselines.

We use the following metrics to evaluate the performance of generated trajectories across PRESTO and other motion planning methods:

- **Success rate:** the percentage of successful trajectories, where the robot completes the trajectory without collisions. Higher is better.
- **Collision rate:** the average fraction of colliding segments in each trajectory. This metric reflects the likelihood of each joint configuration being collision-free, even if there are collisions in the overall trajectory. Lower is better.
- **Penetration depth:** the average maximum penetration depth in each trajectory. This metric evaluates the deviation from a collision-free trajectory. Lower is better, as it indicates easier trajectory optimization.

To systematically evaluate performance changes across different computation budgets, we vary the number of optimization iterations during the post-processing phase. The Bi-RRT baseline was evaluated on an AMD Ryzen 9 5900X and all other baselines were evaluated on an NVIDIA A5000.

### B. Quantitative Evaluation

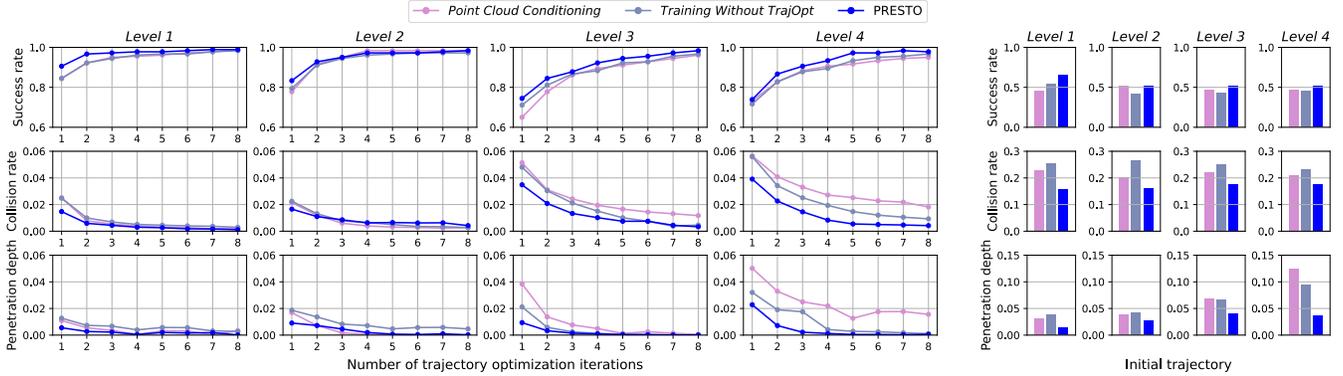
In our experiments, we seek to evaluate the following claims:

- *Claim 1.* Diffusion models using key-configuration-based representations generalize better to unseen environments than those using point-cloud-based representations.
- *Claim 2.* Incorporating trajectory optimization costs in diffusion-model training enables the model to better learn task constraints like collision avoidance, compared to using only the original reconstruction-based objective.
- *Claim 3.* By seeding trajectories for planners, our method achieves better computational efficiency compared to pure planners and higher-quality trajectories compared to pure learning-based approaches.

To validate our claims, we compare our model’s performance against the following baselines, as presented in Figure 3 (bottom).

- **Bi-RRT:** a pure planner from LaValle et al. [23]. Bi-RRT searches bidirectionally by building random trees from both start and goal configurations to find collision-free trajectories in an unknown C-space. Though probabilistically complete, it empirically suffers from slow search in narrow-passage domains. Since Bi-RRT’s success rate depends on the provided time, we evaluate its performance across different search timeouts.
- **TrajOpt:** a pure planner from Schulman et al. [33] which optimizes trajectories using a hinge penalty for collisions and configuration distances. The trajectory optimization scheme in TrajOpt is the same as PRESTO, without the initial seed from our diffusion model. We use a GPU-accelerated implementation from cuRobo [35] and control the compute budget by adjusting optimization iterations.
- **SceneDiffuser:** a learning algorithm by Huang et al. [10] uses a conditional diffusion model for trajectory planning through sampling. Unlike PRESTO, this baseline conditions on point cloud inputs encoded by Point Transformer [40] instead of the C-space representation. We use the author’s original network implementation, trained on our dataset.
- **Motion Planning Diffusion (MPD):** a baseline from Carvalho et al. [4] uses a diffusion model for trajectory planning through sampling. Unlike PRESTO, MPD employs unconditional diffusion models and relies on sampling guidance for trajectory constraints like collision avoidance or connecting start and goal configurations. We adapt their network to our setup and training dataset.

*a) Comparison to pure learning algorithms:* We first consider pure learning algorithms SceneDiffuser and MPD, which do not use a key-configuration-based environment representation (*Claim 1*) or the motion-planning-based objective for training diffusion models (*Claim 2*). To this end, we evaluate each baseline’s diffusion model performance *without* trajectory optimization post-processing, as shown by the large black, yellow, and purple dots in Figure 3,



**Fig. 4: Ablation study results.** We report the success rate (%), collision rate (%), and penetration depth (m) averaged across 180 problems for PRESTO and the self-variant baselines. (Left) We show performance changes with varying post-processing iterations. (Right) We present performance of trajectories directly generated by the diffusion models, without post-processing.

which correspond to PRESTO, SceneDiffuser, and MPD. PRESTO consistently outperforms both SceneDiffuser and MPD across all levels. For example, in *Level 3*, PRESTO has a 52.2% success rate, while SceneDiffuser and MPD only achieve 0.6% and 12.2%, respectively.

*b) Comparison to pure planners:* Compared to Bi-RRT, PRESTO uses diffusion-learned trajectory priors to generate collision-free trajectories more efficiently, especially in narrow passages. In *Level 4*, PRESTO achieves a 90% success rate in 1.0 seconds, compared to 2.3 seconds for Bi-RRT. Furthermore, as environment complexity increases, Bi-RRT struggles to find valid segments, as the success rate gap widens from 97.8% vs. 70.6% in *Level 1* to 90.6% vs. 46.7% in *Level 4* with a 1-second compute budget. Next, we consider TrajOpt, an optimization-based method. Despite PRESTO’s computational overhead for running the diffusion model, its high-quality initial trajectories lead to faster convergence in complex domains (*Claim 3*). For example, in *Level 2*, PRESTO achieves a 97.2% success rate in 1.0 seconds, despite the initial overhead of 0.2 seconds. On the other hand, TrajOpt’s success rate is under 60% in 1.0 seconds. The success rate gap with a 1-second compute budget grows from 26.7% in *Level 1* to 37.3% in *Level 4*, demonstrating PRESTO’s effectiveness in complex environments.

### C. Ablation Studies

To analyze the impact of our contributions and discuss the claims from Section V-B, we conduct ablation studies using variants of our method, with results shown in Figure 4. Additional studies are available on our project website.

- **Point-Cloud Conditioning:** To validate *Claim 1*, we train a variant of PRESTO conditioned on workspace point-clouds instead of key configurations. We use a Patch-based transformer [37] to encode the point clouds. The rest of the architecture remains unchanged.
- **Training Without TrajOpt:** To validate *Claim 2*, we train a variant of PRESTO without motion-planning costs (*Collision Loss* and *Distance Loss*). The rest of the architecture remains unchanged.

*a) Generalization of key-configuration representation (Claim 1):* Compared to PRESTO, *Point-Cloud Conditioning*

exhibits performance degradation across problem levels and post-processing iterations: collision rates and penetration depth remain higher, and worsen with increased problem complexity. For instance, in *Level 1-2*, PRESTO outperforms *Point-Cloud Conditioning* by 1.4% in success rate, 0.3% in collision rate, and 0.001m in penetration depth. The gaps increase to 3.6%, 1.4%, and 0.013m in *Level 3-4*. This highlights the advantage of using C-space representations over point-cloud-based conditioning in complex scenes.

*b) Efficacy of training with TrajOpt loss (Claim 2):* Compared to PRESTO, *Training Without TrajOpt* exhibits performance degradation across all levels. Though less severe than *Point-Cloud Conditioning*, the trend is consistent: for example, in *Level 1-2*, PRESTO outperforms *Training Without TrajOpt* by an average of 1.81% in success rate, 0.2% in collision rate, and 0.005m in penetration depth. The gaps increase to 2.7% in success rate, 0.7% in collision rate, and 0.004m in penetration depth in *Level 3-4*. This shows that incorporating TrajOpt costs for training diffusion models leads to improved trajectory quality across various domains.

*c) Efficacy of post-processing (Claim 3):* We observe that applying trajectory optimization during post-processing improves performance across all levels. Additionally, we validate that the success of PRESTO is largely due to the high-quality (nearly collision-free) initial trajectories obtained from our diffusion model. As shown in Figure 4 (right), PRESTO in *Level 4* outperforms *Point-Cloud Conditioning* with a much smaller penetration depth (0.037 m vs. 0.123 m), despite similar initial success rates (51.1% vs. 47.2%), leading to faster convergence during trajectory optimization.

## VI. CONCLUSION

We present PRESTO, a learning-guided motion planning framework that integrates diffusion-based trajectory sampling with post-processing trajectory optimization. By incorporating C-space environment representations based on key configurations, along with TrajOpt-inspired training objective for training diffusion models, our framework can efficiently generate collision-free trajectories in unseen environments. In simulated experiments, we demonstrate the efficacy of our framework compared to prior diffusion-based approaches in planning, as well as conventional motion-planning methods.

## REFERENCES

- [1] 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.
- [2] O. Arslan and P. Tsiotras, "Machine learning guided exploration for sampling-based motion planning algorithms," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.
- [3] T. D. Barfoot, C. H. Tong, and S. Särkkä, "Batch continuous-time trajectory estimation as exactly sparse gaussian process regression," in *Robotics: Science and Systems*, 2014.
- [4] J. Carvalho, A. T. Le, M. Baierl, D. Koert, and J. Peters, "Motion planning diffusion: Learning and planning of robot motions with diffusion models," *arXiv preprint arXiv:2308.01557*, 2023.
- [5] M. Danielczuk, A. Mousavian, C. Eppner, and D. Fox, "Object rearrangement using learned implicit collision functions," in *IEEE International Conference on Robotics and Automation*, 2021.
- [6] N. Das and M. Yip, "Learning-based proxy collision detection for robot motion planning applications," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1096–1114, 2020.
- [7] *Franka Robotics*, <https://franka.de/>.
- [8] G. Giannone, A. Srivastava, O. Winther, and F. Ahmed, "Aligning optimization trajectories with diffusion models for constrained design generation," *arXiv preprint arXiv:2305.18470*, 2023.
- [9] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Advances in Neural Information Processing Systems*, 2020.
- [10] S. Huang *et al.*, "Diffusion-based generation, optimization, and planning in 3d scenes," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.
- [11] J. Huh and D. D. Lee, "Learning high-dimensional mixture models for fast collision detection in rapidly-exploring random trees," in *IEEE International Conference on Robotics and Automation*, 2016.
- [12] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *IEEE International Conference on Robotics and Automation*, 2018.
- [13] T. F. Iversen and L.-P. Ellekilde, "Kernel density estimation based self-learning sampling strategy for motion planning of repetitive tasks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016.
- [14] M. Janner, Y. Du, J. Tenenbaum, and S. Levine, "Planning with diffusion for flexible behavior synthesis," in *International Conference on Machine Learning*, 2022.
- [15] N. Jetchev and M. Toussaint, "Fast motion planning from experience: Trajectory prediction for speeding up movement generation," *Autonomous Robots*, vol. 34, pp. 111–127, 2013.
- [16] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
- [17] J. C. Kew, B. Ichter, M. Bandari, T.-W. E. Lee, and A. Faust, "Neural collision clearance estimator for batched motion planning," in *Algorithmic Foundations of Robotics XIV: Proceedings of the Fourteenth Workshop on the Algorithmic Foundations of Robotics 14*, Springer, 2021, pp. 73–89.
- [18] 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 Conference on Artificial Intelligence*, 2018.
- [19] B. Kim, L. P. Kaelbling, and T. Lozano-Pérez, "Adversarial actor-critic method for task and motion planning problems using planning experience," in *AAAI Conference on Artificial Intelligence*, 2019.
- [20] R. Kumar, A. Mandalika, S. Choudhury, and S. Srinivasa, "Lego: Leveraging experience in roadmap generation for sampling-based planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019.
- [21] Y.-L. Kuo, A. Barbu, and B. Katz, "Deep sequential models for sampling-based planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018.
- [22] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.
- [23] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and computational robotics*, pp. 303–307, 2001.
- [24] A. Murali, A. Mousavian, C. Eppner, A. Fishman, and D. Fox, "Cabinet: Scaling neural collision detection for object rearrangement with procedural scene generation," in *IEEE International Conference on Robotics and Automation*, May 2023.
- [25] Y. Pang, W. Wang, F. E. Tay, W. Liu, Y. Tian, and L. Yuan, "Masked autoencoders for point cloud self-supervised learning," in *European Conference on Computer Vision*, 2022.
- [26] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019.
- [27] W. Peebles and S. Xie, "Scalable diffusion models with transformers," *arXiv preprint arXiv:2212.09748*, 2022.
- [28] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, "Motion planning networks: Bridging the gap between learning-based and classical motion planners," *IEEE Transactions on Robotics*, vol. 37, no. 1, pp. 48–66, 2020.
- [29] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *IEEE international conference on robotics and automation*, 2009.
- [30] K. Saha *et al.*, "EDMP: Ensemble-of-costs-guided diffusion for motion planning," in *2024 IEEE International Conference on Robotics and Automation*, 2024.
- [31] C. Saharia *et al.*, "Photorealistic text-to-image diffusion models with deep language understanding," in *Advances in Neural Information Processing Systems*, 2022.
- [32] T. Salimans and J. Ho, "Progressive distillation for fast sampling of diffusion models," in *International Conference on Learning Representations*, 2022.
- [33] J. Schulman *et al.*, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.
- [34] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," *arXiv preprint arXiv:2010.02502*, 2020.
- [35] B. Sundaralingam *et al.*, "cuRobo: Parallelized collision-free minimum-jerk robot motion generation," *arXiv preprint arXiv:2310.17274*, 2023.
- [36] C. Yu and S. Gao, "Reducing collision checking for sampling-based motion planning using graph neural networks," in *Advances in Neural Information Processing Systems*, 2021.
- [37] X. Yu, L. Tang, Y. Rao, T. Huang, J. Zhou, and J. Lu, "Pointbert: Pre-training 3d point cloud transformers with masked point modeling," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2022.
- [38] C. Zhang, J. Huh, and D. D. Lee, "Learning implicit sampling distributions for motion planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018.
- [39] M. Zhang *et al.*, "Motiondiffuse: Text-driven human motion generation with diffusion model," *arXiv preprint arXiv:2208.15001*, 2022.
- [40] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *IEEE/CVF international conference on computer vision*, 2021.
- [41] Y. Zhi, N. Das, and M. Yip, "Diffco: Autodifferentiable proxy collision detection with multiclass labels for safety-aware trajectory optimization," *IEEE Transactions on Robotics*, vol. 38, no. 5, pp. 2668–2685, 2022.

### A. Incorporating Guidance During Sampling

In an unconditional diffusion model, test-time guidance [14, 39] constrains trajectories to specific environments and start/goal configurations. While prior works [4, 10] rely on test-time guidance for collision avoidance and endpoint constraints, we use only conditional diffusion models and trajectory optimization for strict constraint satisfaction. Here, we also include an ablation study on the complementary use of guidance steps during sampling to enhance motion planning performance.

*a) Guidance Function Implementation:* As in MPD [4], the guidance function includes collision and smoothness costs (same as Section IV-B). Collision costs are computed with cuRobo [35], and smoothness costs use a Gaussian Process prior [3, 4]. To ensure stability during guidance, we first smooth the sampled trajectory with a Gaussian kernel ( $\sigma = 4.0$ ) before computing costs, allowing collision-cost gradients to affect neighboring points. We then compute the clamped collision cost ( $d_{\max} = 0.1$ ) and the smoothness cost, summing them as  $k_{\text{smooth}}c_{\text{smooth}} + k_{\text{coll}}c_{\text{coll}}$ , with  $k_{\text{smooth}} = 1e - 9$  and  $k_{\text{coll}} = 1e - 2$ . Gradients are computed with PyTorch [26], clamped ( $g_{\max} = 1.0$ ) to prevent erratic updates, zeroed at endpoints, and added directly to the trajectory.

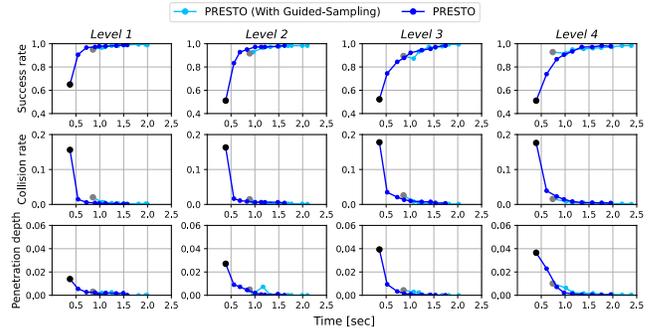
*b) Results:* Figure 6 compares our model with a variant that includes guidance steps during denoising iterations. Overall, guided-sampling enhances the quality of initial trajectories (black dots represent PRESTO without guidance, and gray dots represent PRESTO with guidance). For example, the success rate of the denoised trajectory before optimization reaches 92.8% in Level 4, compared to 51.1% for PRESTO without guidance. However, incorporating guidance requires gradient evaluations for the costs at each diffusion iteration, resulting in computational overhead. In Level 3, this overhead accumulates to an average of 0.38 seconds, indicating that the added cost of guidance steps may occasionally degrade performance within a given time frame. Despite this, guidance generally improves performance across Levels 1-4 in terms of all three metrics: success rate, collision rate, and penetration depth, given the same

number of trajectory optimization iterations.

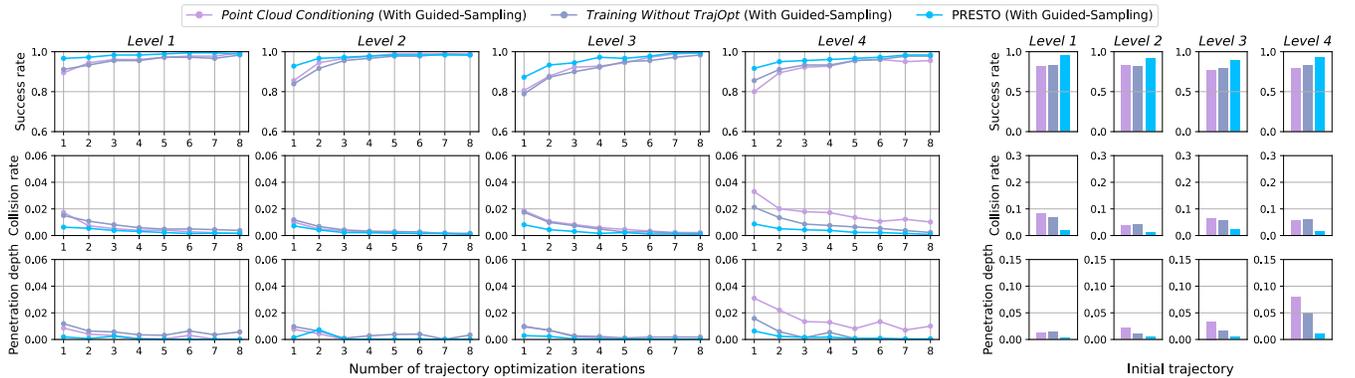
We also report the effects of guidance on variants of our model in Figure 5. Performance across all baselines improves when guidance steps are applied compared to the original results in Figure 4. Notably, the success rate gap between PRESTO and its variants widens with the application of guidance. For instance, across Level 1-4, the gap between PRESTO and the closest baseline (*Point-Cloud Conditioning*) increases from 2.4% to 4.0%. As PRESTO generates higher-quality initial trajectories with smaller penetration depths, spurious collisions are resolved with just a few guidance steps, leading to greater success rate gains compared to the ablations of PRESTO.

### B. Point Cloud Encoder Architecture

For our ablation with point-cloud inputs (Section V-C), we design the point-cloud encoder based on recent patch-based transformers [25, 37]. We divide the  $\mathbb{R}^{1024 \times 3}$  point cloud into 8 patches using farthest-point sampling and k-nearest neighbors ( $k = 128$ ). Each patch is normalized, flattened, and projected into shape embeddings via a 3-layer MLP with GeLU and Layer Normalization. Positional embeddings, computed from patch centers using a 2-layer MLP, are added before processing with a 4-layer transformer to extract geometric features, which serve as additional input tokens for the DiT in the diffusion model.



**Fig. 6: Results with guided-sampling.** We report the success rate (%), collision rate (%), and penetration depth (m) averaged across 180 problems for PRESTO and the self-variant baselines. Black dots represent PRESTO without post-processing, while gray dots represent PRESTO with Guided Sampling, also without post-processing.



**Fig. 5: Ablation studies with guided-sampling.** We report the success rate (%), collision rate (%), and penetration depth (m) averaged across 180 problems for PRESTO and the self-variant baselines with guided-sampling. (Left) We show performance changes with varying post-processing iterations. (Right) We present performance of trajectories directly generated by the diffusion models, without post-processing.