PRESTO: Fast Motion Planning Using Diffusion Models Based on Key-Configuration Environment Representation

Mingyo Seo[1*], Yoonyoung Cho[2*], Yoonchang Sung[1], Peter Stone[1,3], Yuke Zhu[1†], and Beomjoon Kim[2†]

***Abstract*— We introduce a learning-guided motion planning framework that generates seed trajectories using a diffusion model for trajectory optimization. Given a workspace, our method approximates the configuration space (C-space) obstacles through an environment representation consisting of a sparse set of task-related key configurations, which is then used as a conditioning input to the diffusion model. The diffusion model integrates regularization terms that encourage smooth, collision-free trajectories during training, and trajectory optimization refines the generated seed trajectories to correct any colliding segments. Our experimental results demonstrate that high-quality trajectory priors, learned through our C-space-grounded diffusion model, enable the efficient generation of collision-free trajectories in narrow-passage environments, outperforming previous 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 [1–3] or optimization-based methods [4, 5] 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 spaces and small volume of solutions. While optimization-based methods can serve as 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 [6–8]. By casting motion planning as sampling from a learned distribution, these models can efficiently generate trajectories within a consistent computational 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 the workspace as input to neural networks instead of the C-space. Instead, we propose representing the environment in terms of key configurations [9], a sparse set of task-related configurations from prior motion planning data. The resulting model no longer needs to learn a generalizable mapping between workspace and C-space obstacle representations, improving generalization and reducing training complexity.

[1]The University of Texas at Austin,
[2]Korea Advanced Institute of Science and Technology, [3]Sony AI,
*Equal contribution, †Equal advising.
Correspondance: `beomjoon.kim@kaist.ac.kr`

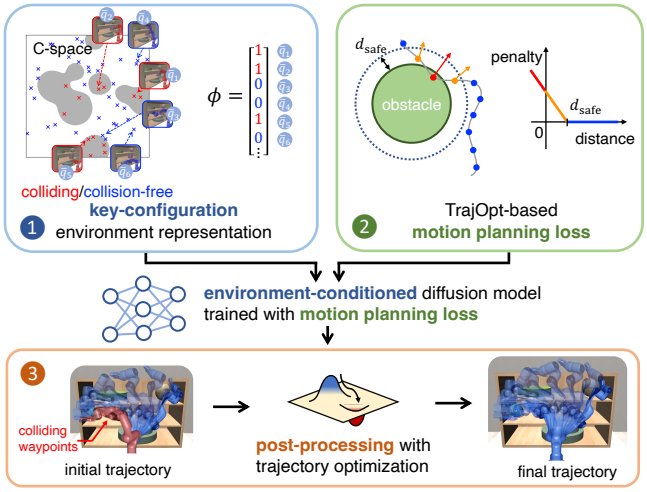

Fig. 1: Overview of PRESTO. PRESTO aims to generate collision-free trajectories in complex, unseen C-spaces. First, we approximate these C-spaces using key configurations from prior data and generate trajectories based on this representation. A conditional diffusion model, trained with a motion planning loss, provides initial solutions that are subsequently refined through trajectory optimization.

Another challenge is designing a training objective for generative models tailored to motion planning. Existing diffusion models for motion planning use DDPM-based losses [6, 8, 10] that focus on reconstruction quality [11]. However, this reconstruction objective does not account for underlying task constraints, resulting in degraded performance on tasks that require precise outputs and complex constraints [12]. To overcome this, we incorporate TrajOpt-inspired motion-planning costs [5] directly into the training pipeline of a diffusion model, minimizing trajectory-optimization costs associated with collision avoidance and trajectory smoothness. As a result, the model learns to generate smooth and collision-free trajectories. Further, to ensure that generated trajectories satisfy hard constraints such as collision avoidance, we feed the diffusion model's outputs as initial solutions for subsequent trajectory optimization.

We name our unified 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 a diffusion model that directly integrates motion-planning costs (green block); and 3) a diffusion-based sampling-and-optimization framework for motion planning, where the diffusion model 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

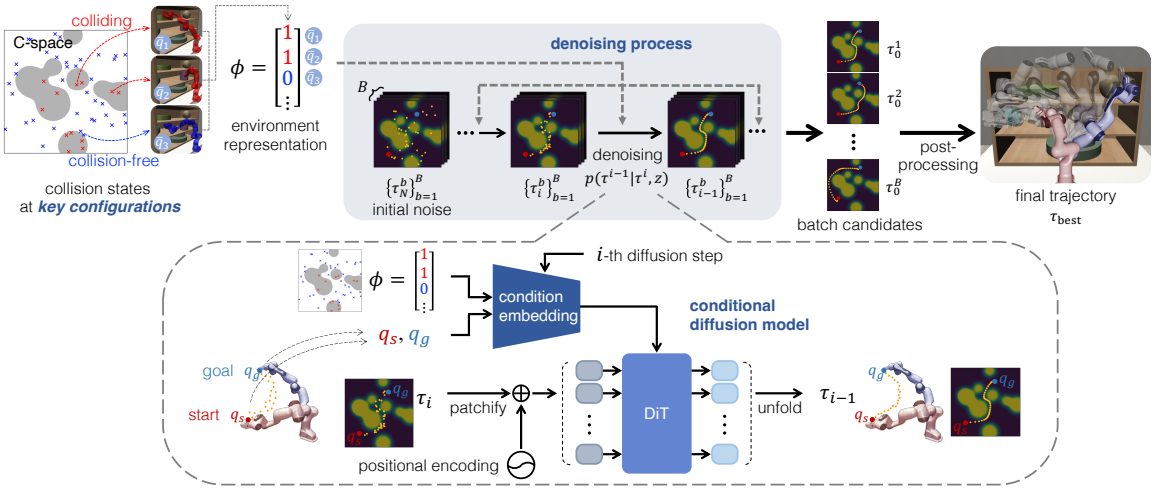

Fig. 2: Trajectory generation pipeline of PRESTO. We obtain the environment representation for an unseen problem by checking the collision states at the key configurations used during training. Using the trained conditional diffusion model, we generate multiple trajectories conditioned on this representation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

shaped and arranged objects. The results show that the synergy between the high-quality trajectory priors generated by our diffusion model and the trajectory optimization post-processing efficiently generates collision-free trajectories in narrow passages within a limited computational budget.

II. Related Work

Several approaches learn to guide motion planning by complementing motion planners with learned models for collision checking [13–18] or for sampling promising configurations [19–26].

A series of work has focused on learning collision checkers to expedite motion planning. One body of work progressively learns collision models in a given environment as Gaussian mixtures [13] or kernel perceptrons [14, 17], while others use deep neural network models trained on large offline datasets. Kew et al. [16] train a neural network to estimate the clearance of robot configurations in a fixed environment. Danielczuk et al. [15] 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. [18] extend neural collision detectors to a partially observable setting. Our approach can integrate with these collision-checking methods to further enhance planning speed.

Another line of work focuses on sampling promising configurations. Some methods derive explicit distributions via kernel density estimation [19, 20], while others use neural networks, in which sequence-based models such as LSTMs [22] or a rejection-sampling policy [21] learn the distribution of collision-free configurations based on the history of collision states at previously sampled configurations. Alternatively, recent works propose using generative models to govern the sampling process. Qureshi et al. [27] map point-cloud inputs to the next sampling configuration, stochastically applying dropout in the interim layers to generate diverse samples; Yu et al. [25] use Graph Neural Networks to identify promising regions of a roadmap; and Ichter et al. [23] along with Kumar et al. [24] employ Conditional Variational Autoencoders (CVAEs) to sample task-related configurations from latent spaces. While these methods generate individual or sets of joint configurations, recent works propose using diffusion models to learn trajectory sampling distributions to accelerate motion planning.

Diffuser [6] 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 [8, 28], but struggles with large environmental variations due to significant mismatches in learned trajectory distributions. Recent studies [29] address this issue by conditioning diffusion models on environmental conditions. Notably, Huang et al. [7] and Xian et al. [30] condition on point-cloud representations to sample motion plans across diverse environments. However, these models still face challenges in generalizing to a new C-space because they rely on workspace inputs, even though motion planning occurs in the C-space.

Prior work explores alternative representations for motion planning, focusing on C-space grounded approaches [9, 31]. These methods approximate complex C-spaces 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. [9], 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 $\tau = (q_0, q_1, \ldots, q_T)$. 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 that we are provided with a

dataset $\mathcal{D} = \{\mathcal{D}_m\}_{m=1}^{M}$ obtained from solving M 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 τ, 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 that incorporates motion-planning costs to guide the model toward smooth and collision-free trajectories. Finally, we feed the trajectories 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 dataset. We denote the set of key configurations as $\{\overline{q}^k\}_{k=1}^{K}$, where the number of key configurations K determines the resolution of the environment's C-space approximation[1]. For each motion planning problem, we compute the environment representation $\phi \in \{0,1\}^K$ as a binary vector that specifies the collision states of each key configuration. In our setups, we use $K = 1025$.

Algorithm 1 describes our procedure for generating key configurations, which is a modified version of the algorithm originally proposed by Kim et al. [26]. It takes the dataset $\mathcal{D}$ and the hyperparameters $d_q^{\min}, d_x^{\min}, c, K$. Here, $d_q^{\min}$ denotes the minimum C-space distance between key configurations, $d_x^{\min}$ represents the minimum workspace distance for end-effector tips, and c is the bound on the proportion of environments where a key configuration is occupied.

[1]A larger K increases the resolution of the C-space approximation, but it also raises the computational overhead at query time.

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

We initialize the key configuration set as an empty set (line 1) and sample new configurations until the target size is reached (lines 2-11). At each step, a configuration is uniformly sampled from $\mathcal{D}$ (lines 3-4) and filtered based on three conditions (lines 5-10). To avoid duplicates, we ensure the new configuration is sufficiently distant from existing ones in both C-space and workspace (lines 5-6), while also limiting the proportion of collision states across different environments to prioritize informative configurations[2] (line 7). If all criteria are met, the configuration is added to the buffer (lines 8-10). This process generates key configurations that effectively capture task-relevant C-space regions for motion planning.

B. Training Conditional Diffusion Models

a) Model Architecture: Figure 2 (bottom) illustrates our model architecture, which is based on the Diffusion Transformer (DiT) [32]. Our model takes as inputs the current diffusion step i, the noisy trajectory at the i-th step τ_i, the environment representation ϕ, and the start and goal joint configurations q_s and q_g. We use v-prediction [33] during inference to enhance sample quality [11].

To process the inputs, the trajectory τ_i is first patchified and tokenized by an MLP, as in DiT [32]. The diffusion step i and the start and goal configurations q_s and q_g are mapped to high-dimensional frequency embeddings [32] to capture small changes. The embedded trajectory patches are given as input tokens to the transformer, while i, ϕ, q_s, and q_g are incorporated as conditioning inputs to align sampled trajectories with the current scene and endpoint constraints.

The DiT comprises six transformer blocks to process the trajectories, where each block incorporates Adaptive Layer Normalization (AdaLN) [32] that transforms the output features based on the conditioning inputs. In AdaLN, a separate MLP maps conditioning inputs to the transform parameters as $\gamma, g, b = \text{MLP}(i, \phi, q_s, q_g)$, applied to the output x 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 allows the model to adjust its output based on the current scene and the diffusion iteration.

b) Training with Motion-Planning Costs: Our training objective includes three terms: *Diffusion Loss*, *Collision Loss*, and *Smoothing Loss*. While *Diffusion Loss* implements the standard reconstruction objective used 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 [5], this term encourages the model to generate trajectories that maintain a safe distance from objects and other links. It is defined as

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

[2]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 .

where $|\cdot|^+ = \max(\cdot, 0)$ and $\text{sd}(\cdot) = \text{dist}(\cdot) - \text{penetration}(\cdot)$ with $d_{\text{safe}} = 0.01$ m as the safety margin of 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 accounts for the collision between the i-th link $\mathcal{A}_i$ and the j-th obstacle $\mathcal{O}_j$, while the second term denotes self-collision among different 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 $w_3 = 0.005$. While the model is not highly sensitive to these values, w_2 is kept small for stable training.

C. Trajectory Generation

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

We detail our trajectory generation in Algorithm 2. First, we compute the environment representation ϕ by checking the collision states of the key configurations $\bar{q}$ (line 1). Next, we initialize the denoising process with a batch of random noise τ_N sampled from an isotropic Gaussian distribution (line 2). At each of the N iterations, we predict a denoised trajectory τ_{i-1} from τ_i, conditioned on ϕ and the current step i (line 5), and then apply endpoint constraints to ensure connectivity between the start q_s and the goal q_g, as in Diffuser [6] (line 6). The resulting trajectories τ_{seed} provide initialization for post-processing (line 9). In our experiments, we denoise $B = 4$ trajectories over $N = 64$ iterations.

In the post-processing phase, we refine the sampled trajectories using a fixed number of trajectory optimization iterations [5] to address potential collisions (line 10). To accelerate this process, we employ cuRobo [35], which can batch-process the trajectories while eliminating data exchange across devices. Afterward, we select the trajectory with the fewest collisions (line 11).

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 τ_{best}
// Computing environment representation
1: $\phi = \texttt{CheckCollision}(\mathcal{G}, \{\bar{q}^k\}_{k=1}^K)$
// Batched denoising with reverse process
2: $\{\tau_N^b \sim \mathcal{N}(\mathbf{0}, \mathbf{I})\}_{b=1}^B$ ▷ Sample a batch of initial trajectories
3: **parallel for** $b = 1, \ldots, B$ **do**
4: **for** $i = N, \ldots, 1$ **do**
5: $\tau_{i-1}^b \sim \mathcal{N}(\mu_\theta(\tau_i^b, \phi, i), \sigma_i)$
6: $q_0 \leftarrow q_s$, $q_T \leftarrow q_g$ ▷ Apply endpoint constraints q_s, q_g
7: **end for**
8: **end parallel**
9: $\tau_{\text{seed}} = \{\tau_0^b\}_{b=1}^B$ ▷ Batch-sized sampled trajectories
// Post-processing
10: $\tau_{\text{opt}} = \texttt{TrajectoryOptimization}(\tau_{\text{seed}})$
11: $\tau_{\text{best}} = \texttt{BestTrajectory}(\tau_{\text{opt}})$
12: **return** τ_{best}

V. Experiments

A. Experimental Setup

We evaluate our method on a motion planning task in which the Franka Emika *Panda* robot arm [36] traverses a 3-tier shelf with various objects in simulation (Figure 3, top).

a) Training Setup: Our training domain consists of 5,000 environments, where 1-6 objects (cuboid, cylinder, sphere) are placed in random poses within each shelf slot. For each scene, we first sample collision-free initial and target joint positions within the workspace and then generate motion plans via cuRobo [35]. We collect a total of 50,000 environment-trajectory pairs (trajectory length $T = 1000$) annotated with key-configuration labels as in Algorithm 1.

b) Evaluation Setup: While the evaluation domain is generated using a similar procedure, we partition the dataset into four difficulty levels, each consisting of 180 problems. This helps assess the performance of PRESTO and the baselines on unseen 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 one object for *Level 2* and two 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.

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

- **Success rate:** the percentage of collision-free trajectories within the batch. Higher is better.
- **Collision rate:** the average fraction of colliding segments in each trajectory. This metric reflects the likelihood that each joint configuration is collision-free, even if collisions occur elsewhere in the trajectory. Lower is better.
- **Penetration depth:** the average maximum penetration depth in each trajectory. This metric quantifies the severity of collision, measuring worst-case deviation from a collision-free trajectory. Lower is better.

To systematically evaluate how performance changes across different computational budgets, we vary the number of optimization iterations during post-processing. The Bi-RRT

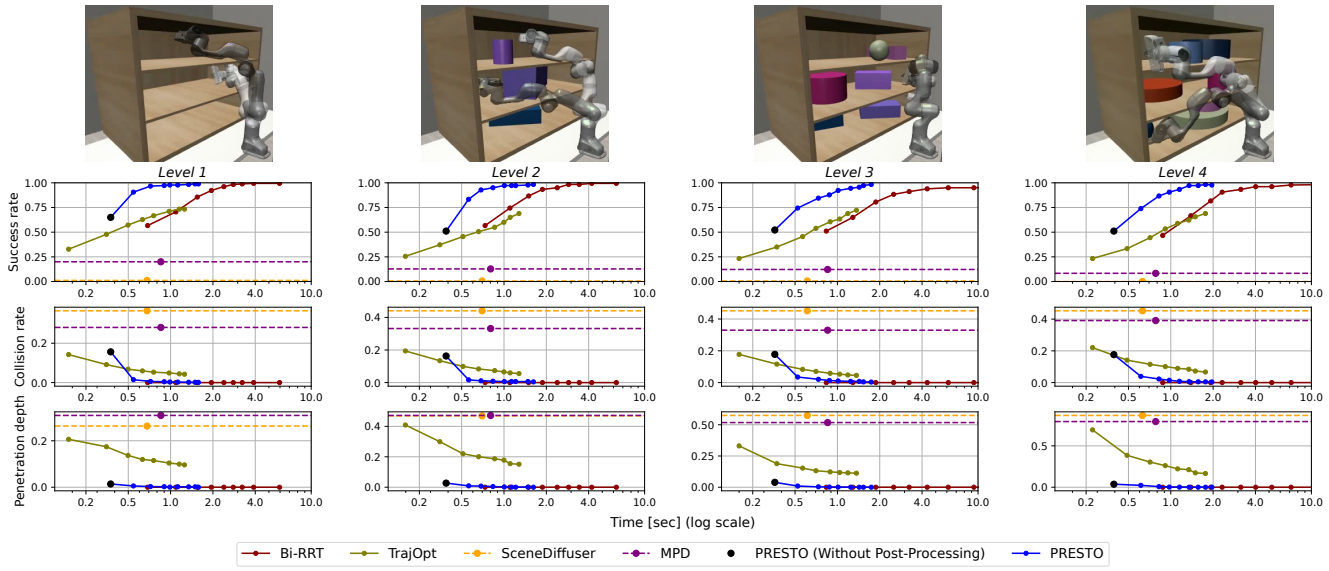

Fig. 3: Main results. We report the success rate (%), the collision rate (%), and the penetration depth (m) across 180 problems. (**Top**) The evaluation environments feature consistent 3-tier shelf fixtures with randomized object positions that vary across levels. (**Bottom**) We show PRESTO's performance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

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 evaluate the following claims: (*Claim 1*) Diffusion models using key-configuration representations generalize better to unseen environments than those using point-cloud representations. (*Claim 2*) Incorporating motion-planning costs into diffusion-model training enables models to better learn task constraints like collision avoidance than when using only the reconstruction-based objective. (*Claim 3*) By seeding trajectories, our method outperforms pure planners in computational efficiency and learning-based approaches in trajectory quality.

To validate our claims, we compare our model's performance against the following baselines, representative of each approach, as presented in Figure 3 (bottom).

- **Bi-RRT:** a pure planner based on LaValle et al. [2], specifically RRT-connect [37]. Bi-RRT searches bidirectionally by growing random trees from both start and goal configurations to find collision-free paths in an unknown C-space. Although it is probabilistically complete, it empirically struggles with slow searches in narrow passages. Since its success rate depends on the provided time, we evaluate its performance across different search timeouts.
- **TrajOpt:** a pure planner from Schulman et al. [5] that optimizes trajectories using a hinge penalty for collisions and configuration distances. The optimization scheme in TrajOpt is the same as in PRESTO, without the initial seed from our diffusion model. We use a GPU-accelerated implementation from cuRobo [35] and control the computational budget by adjusting optimization iterations.
- **SceneDiffuser:** a baseline from Huang et al. [7] uses a conditional diffusion model for trajectory planning through sampling. Unlike PRESTO, this baseline conditions on point-cloud inputs encoded by Point Transformer [38] instead of a 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. [8] uses a diffusion model for trajectory planning through sampling. Unlike PRESTO, MPD employs unconditional diffusion models and relies on sampling guidance for trajectory constraints such as 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 consider pure learning algorithms, SceneDiffuser and MPD, which neither use a key-configuration environment representation (*Claim 1*) nor a motion-planning objective for training diffusion models (*Claim 2*). We evaluate each baseline's diffusion model performance *without* trajectory optimization post-processing, as indicated by the large black, yellow, and purple dots in Figure 3 corresponding to PRESTO, SceneDiffuser, and MPD. PRESTO consistently outperforms both across all levels. For example, in *Level 3*, PRESTO achieves a 52.2% success rate, while SceneDiffuser and MPD achieve only 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 second, compared to 2.3 seconds for Bi-RRT. Furthermore, as environment complexity increases, Bi-RRT struggles to find valid segments, widening the success gap from 97.8% vs. 70.6% in *Level 1* to 90.6% vs. 46.7% in *Level 4* under a 1-second computational 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 ex-

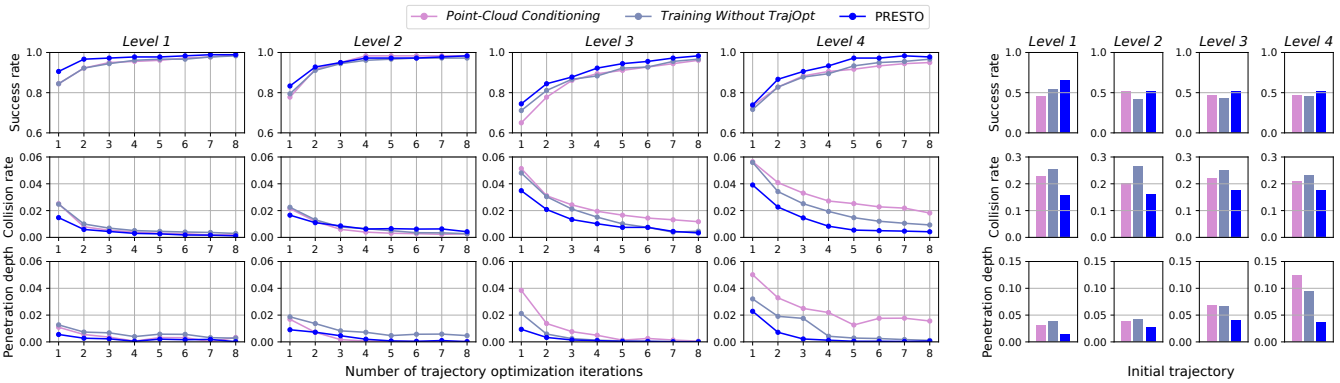

Fig. 4: Ablation study results. We report the success rate (%), the collision rate (%), and the penetration depth (m) averaged across 180 problems for PRESTO and the self-variant baselines. (**Left**) We show performance changes with varying post-processing iterations. (**Right**) We present the performance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

ample, in *Level 2*, PRESTO achieves a 97.2% success rate in 1.0 second despite an initial overhead of 0.2 seconds, while TrajOpt remains below 60% in the same time. The success rate gap with a 1-second computational 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 an equivalent number of workspace point clouds instead of key configurations. We use a patch-based transformer [39] to encode the point clouds, keeping the rest of the architecture 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. In *Level 3-4*, the gaps increase to 3.6%, 1.4%, and 0.013m. This highlights the advantage of using C-space representations over point-cloud-based conditioning in complex scenes.

b) Efficacy of Training with Motion-Planning Costs (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 and 0.2% in collision rate. In *Level 3-4*, the performance gap increases to 2.7% in success rate and 0.7% in collision rate. This shows that incorporating motion-planning costs for training diffusion models improves 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* by achieving 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. Incorporating C-space environment representations based on key configurations and a motion-planning training objective, our framework efficiently generates collision-free trajectories in unseen environments. Simulated experiments demonstrate the efficacy of our framework compared to both diffusion-based planning approaches and conventional motion planning methods. In this work, we assumed known environment geometry for ground-truth collision states at key configurations. Future work could extend the framework to handle partial observability of unseen geometries.

Acknowledgements This work was partially supported by the Institute of Information & Communications Technology Planning & Evaluation (Nos. 2022-0-00311, 2022-0-00612. 2024-00509279, 2019-190075), the National Research Foundation of Korea (No. RS-2024-00359085) funded by the Korean government (MSIT), the National Science Foundation (FRR2145283, EFRI-2318065, FAIN-2019844, NRT-2125858), the Office of Naval Research (N00014-22-1-2204, N00014-24-1-2550, N00014-18-2243), DARPA (TIAMAT program HR0011-24-9-0428, Cooperative Agreement HR00112520004 on Ad Hoc Teamwork), the Army Research Office (W911NF-23-2-0004, W911NF-17-2-0181), Lockheed Martin, and UT Austin's Good Systems grand challenge. Peter Stone serves as the Chief Scientist of Sony AI and receives financial compensation for that role. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

REFERENCES

[1] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[2] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and computational robotics*, pp. 303–307, 2001.

[3] 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.

[4] 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.

[5] 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.

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

[7] S. Huang *et al.*, "Diffusion-based generation, optimization, and planning in 3d scenes," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[8] 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 *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2023.

[9] 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.

[10] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Advances in Neural Information Processing Systems*, 2020.

[11] C. Saharia *et al.*, "Photorealistic text-to-image diffusion models with deep language understanding," in *Advances in Neural Information Processing Systems*, 2022.

[12] G. Giannone, A. Srivastava, O. Winther, and F. Ahmed, "Aligning optimization trajectories with diffusion models for constrained design generation," in *Advances in Neural Information Processing Systems*, 2023.

[13] 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.

[14] 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.

[15] 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.

[16] J. C. Kew, B. Ichter, M. Bandari, T.-W. E. Lee, and A. Faust, "Neural collision clearance estimator for batched motion planning," in *Workshop on the Algorithmic Foundations of Robotics*, 2020.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *IEEE International Conference on Robotics and Automation*, 2018.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] K. Saha *et al.*, "EDMP: Ensemble-of-costs-guided diffusion for motion planning," in *IEEE International Conference on Robotics and Automation*, 2024.

[29] A. Ajay, Y. Du, A. Gupta, J. Tenenbaum, T. Jaakkola, and P. Agrawal, "Is conditional generative modeling all you need for decision-making?" In *International Conference on Learning Representations*, 2023.

[30] Z. Xian, N. Gkanatsios, T. Gervet, T.-W. Ke, and K. Fragkiadaki, "Chaineddiffuser: Unifying trajectory diffusion and keypose prediction for robotic manipulation," in *Conference on Robot Learning*, 2023.

[31] 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.

[32] W. Peebles and S. Xie, "Scalable diffusion models with transformers," in *International Conference on Computer Vision*, 2023.

[33] T. Salimans and J. Ho, "Progressive distillation for fast sampling of diffusion models," in *International Conference on Learning Representations*, 2022.

[34] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," in *International Conference on Learning Representations*, 2021.

[35] B. Sundaralingam *et al.*, "cuRobo: Parallelized collision-free minimum-jerk robot motion generation," *arXiv preprint arXiv:2310.17274*, 2023.

[36] *Franka Robotics*, `https://franka.de/`.

[37] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation*, 2000.

[38] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *IEEE/CVF International Conference on Computer Vision*, 2021.

[39] X. Yu, L. Tang, Y. Rao, T. Huang, J. Zhou, and J. Lu, "Point-bert: Pre-training 3d point cloud transformers with masked point modeling," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

[40] M. Zhang *et al.*, "Motiondiffuse: Text-driven human motion generation with diffusion model," *arXiv preprint arXiv:2208.15001*, 2022.

[41] 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.

[42] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019.

[43] 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.

[44] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," in *Advances in Neural Information Processing Systems*, 2021.

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

APPENDIX

A. Incorporating Guidance During Sampling

In an unconditional diffusion model, test-time guidance [6, 40] constrains trajectories to specific environments and start/goal configurations. While prior works [7, 8] 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: Algorithm 3 describes our procedure for computing cost gradients. As in MPD [8], the guidance function includes collision and smoothness costs as described in Section IV-B. Collision costs are computed using cuRobo [35], while smoothness costs employ a Gaussian Process prior [8, 41]. To ensure stability during guidance, we smooth the sampled trajectory with a Gaussian kernel ($\sigma = 4.0$) before computing costs, allowing collision-cost gradients to affect neighboring points (line 1). We then compute the clamped collision cost ($d_{\texttt{max}} = 0.1$) 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 $k_{\text{coll}} = 1e-2$ (line 2-4). Gradients are computed with PyTorch [42], clamped ($g_{\texttt{max}} = 1.0$) to prevent erratic updates, zeroed at endpoints, and directly added to the trajectory (line 5-6).

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 evaluating cost gradients at each diffusion iteration, resulting in computational overhead. In *Level 3*, this overhead accumulates to an average of 0.38 seconds, indicating that the additional cost of guidance steps may occasionally degrade performance within a given time frame. Despite this, guidance generally improves performance across *Level 1-4* in 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. Consistent with the results in Figure 6, 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.

Algorithm 3 Guidance Step with Cost Gradients.

Require: Trajectory τ, Smoothing Kernel $\mathcal{K}$, Environment $\mathcal{G}$
Ensure: Output trajectory $\tau_{\texttt{guide}}$
1: $\tilde{\tau} = \texttt{Convolve}(\texttt{UnNormalize}(\tau), \mathcal{K})$
// Computing costs
2: $c_{\texttt{coll}} = \texttt{min}(\texttt{CollisionCost}(\tilde{\tau}, \mathcal{G}), d_{\texttt{max}})$
3: $c_{\texttt{smooth}} = \texttt{GPCost}(\tau)$
4: $c_{\texttt{guide}} = k_{\texttt{smooth}}c_{\texttt{smooth}} + k_{\texttt{coll}}c_{\texttt{coll}}$
// Applying guidance gradients
5: $g_{\texttt{guide}} = \texttt{Clamp}(\nabla_{\tau}(c_{\texttt{guide}}), g_{\texttt{max}})$
6: $\tau_{\texttt{guide}} = \tau - g_{\texttt{guide}}$
7: **return** $\tau_{\texttt{guide}}$

B. Computation Time Details

We provide computation time details for PRESTO in Section V and present them in Figure 7. The key-configuration query incurs an average overhead of 0.658 milliseconds, which is negligible compared to trajectory generation in the diffusion model, averaging 0.345 seconds, and trajectory optimization during post-processing, which scales linearly with the number of iterations. The performance improvements of PRESTO, leveraging an environment representation based on key configurations, over the *Point-Cloud Conditioning* baseline in Section V-C demonstrate its effectiveness in enhancing representation quality without compromising computational speed. This makes it particularly well-suited for scenarios with limited computational resources.

C. Point-Cloud Encoder Architecture

For our ablation with point-cloud inputs in Section V-C, we design the point-cloud encoder based on recent patch-based transformers [39, 43]. We divide the $\mathbb{R}^{1024\times3}$ 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.

D. Background

a) Denoising Diffusion Probabilistic Models: The diffusion model involves a forward diffusion process, which starts from a perfect solution (in our case a collision-free trajectory) and gradually injects noise until reaching an isotropic Gaussian distribution, and a reverse diffusion process, which learns to generate data through gradual denoising. Let $i \in \{0, 1, ..., N\}$ denote a denoising step. The data generation process using a diffusion model involves iteratively denoising noise τ_i until reaching a denoised sample τ_0, where τ is a sequence of robot configurations in this work. The reverse diffusion process is given by:

$$p_\theta(\tau_{0:T}) = p(\tau_T)\prod_{i=1}^{N} p_\theta(\tau_{i-1}|\tau_i),$$

where $p_\theta(\tau_{i-1}|\tau_i) = \mathcal{N}\big(\tau_{i-1}; \mu_\theta(\tau_i, i), \Sigma_\theta(\tau_i, i)\big)$, as proven by Ho *et al*. [10] under some mild conditions. Ho *et al*. [10] show that $\mu_\theta(\tau_i, i)$ can be modeled as a function $\epsilon_\theta(\tau_i, i)$,

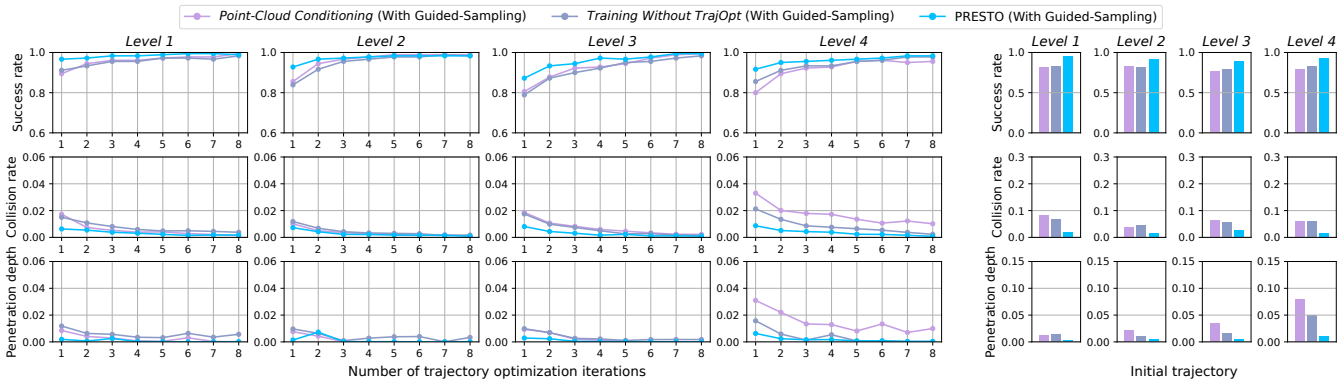

Fig. 5: Ablation studies with guided-sampling. We report the success rate (%), the collision rate (%), and the penetration depth (m) averaged across 180 problems for PRESTO and the self-variant baselines with guided-sampling. (**Left**) We show performance changes with varying post-processing iterations.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

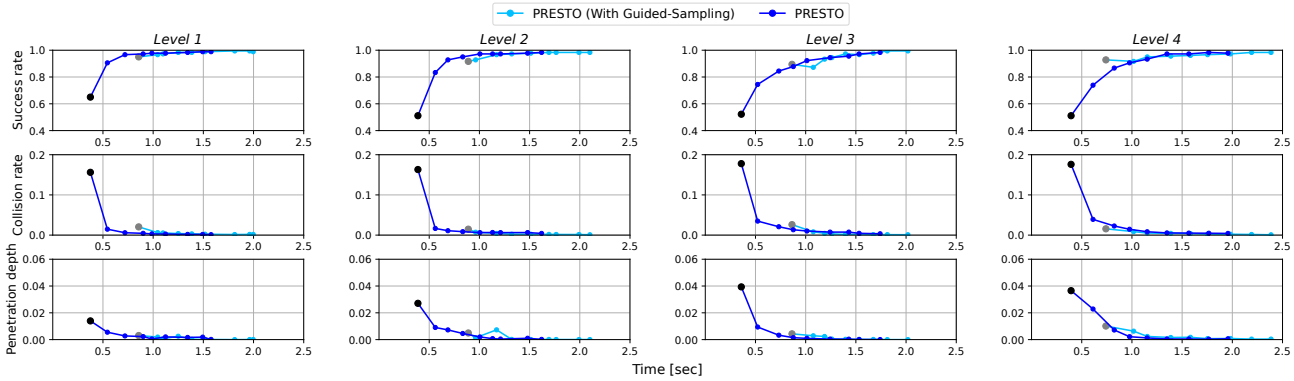

Fig. 6: Results with guided-sampling. We report the success rate (%), the collision rate (%), and the penetration depth (m) averaged across 180 problems for PRESTO and the self-variant baselines. Black dots represent PRESTO without post-processing, while gray dots represent PRESTO with Guided Sampling,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

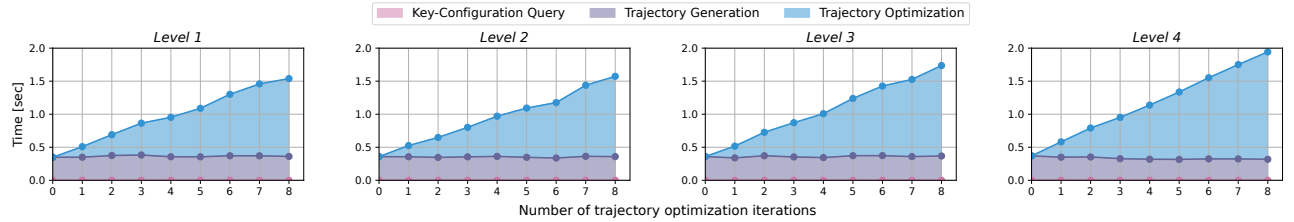

Fig. 7: Breakdown of computation time for PRESTO. We report the average computation time (sec) across 180 problems for PRESTO and provide a breakdown of the time taken for each step: the key-configuration query for environment representation, trajectory generation using the diffusion model, and trajectory optimization during post-processing.

which predicts a noise value from a sample at the i-th denoising step. In addition, they show that the training objective can be simplified $\min_\theta ||\epsilon_\theta(\tau_i, i) - \epsilon_i||^2$, and the diffusion model can be trained by minimizing this term.

b) Conditional Diffusion Models: We convert the unconditional form of $p_\theta(\tau_{i-1}|\tau_i)$ to be conditioned on our environment representation. We adopt the methods proposed by Dhariwal and Nichol [44], whose objective is to generate an image conditioned on the class. Specifically, they demonstrate that the term of the conditional reverse diffusion process can be factorized as follows: $p_{\theta,\phi}(\tau_{i-1}|\tau_i, \phi) \propto p_\theta(\tau_{i-1}|\tau_i)p_\phi(\phi|\tau_{i-1})$, where ϕ is the conditioning variable, which in our case will be an environment representation. Under some mild conditions, they show that after applying the logarithm, we get $\log\left(p_\theta(\tau_{i-1}|\tau_i)\ p_\phi(\phi|\tau_{i-1})\right) \approx \log p(u) +$ const, where $u \sim \mathcal{N}\left(\tau_{i-1}; \mu_\theta(\tau_i, i) + g\Sigma_\theta(\tau_i, i),\ \Sigma_\theta(\tau_i, i)\right)$ and $g = \bigtriangledown_{\tau_{i-1}} \log p_\phi(\phi|\tau_{i-1})|_{\tau_{i-1}=\mu_\theta(\tau_i,i)}$. Notice that the mean parameter of the Gaussian term in u includes an additional component, $g\Sigma_\theta(\tau_i, i)$, which is not a part of the mean parameter of $p_\theta(\tau_{i-1}|\tau_i)$. This mathematical derivation implies that the diffusion model can generate a sample conditioned on ϕ by the gradients g. Based on this, we particularly use a classifier-free guidance diffusion model to implement our motion generation. It is known that conditional diffusion steps can be run by incorporating the probabilities from both a conditional and an unconditional diffusion model [45], as $\bigtriangledown \log p_\phi(\phi|\tau) = \bigtriangledown \log p_\phi(\tau|\phi) - \bigtriangledown \log p_\phi(\tau)$. Therefore, by plugging into the conditional probabilities, conditional guidance can be trained together with the diffusion models, eliminating the need for a separate classifier.