Adaptive Diffusion Constrained Sampling for Bimanual Robot Manipulation

Haolei Tong[1, *], Yuezhe Zhang[1, *], Sophie Lueth[1], Georgia Chalvatzaki[1, 2, 3]
[1]TU Darmstadt, [2]Hessian.AI, [3]Robotics Institute Germany
[*]Equal Conrtibution
adaptive-diffusion-constrained-sampling.github.io

Abstract: Coordinated multi-arm manipulation requires satisfying multiple simultaneous geometric constraints across high-dimensional configuration spaces, which poses a significant challenge for traditional planning and control methods. In this work, we propose Adaptive Diffusion Constrained Sampling (ADCS), a generative framework that flexibly integrates both equality (e.g., relative and absolute pose constraints) and structured inequality constraints (e.g., proximity to object surfaces) into an energy-based diffusion model. Equality constraints are modeled using dedicated energy networks trained on pose differences in Lie algebra space, while inequality constraints are represented via Signed Distance Functions (SDFs) and encoded into learned constraint embeddings, allowing the model to reason about complex spatial regions. A key innovation of our method is a Transformer-based architecture that learns to weight constraint-specific energy functions at inference time, enabling flexible and context-aware constraint integration. Moreover, we adopt a two-stage batch-wise constrained sampling strategy that improves precision and sample diversity by combining Langevin dynamics with resampling and density-aware re-weighting. Experimental results on dual-arm manipulation tasks show that ADCS significantly improves sample diversity and generalization across settings demanding precise coordination and adaptive constraint handling.

Keywords: Diffusion Models, Bimanual Manipulation, Constrained Sampling

1 Introduction

Robots operating in human environments must be capable of performing increasingly complex tasks that involve interaction with diverse objects, unstructured scenes, and coordination across multiple end-effectors. As tasks grow in complexity [1], such as moving large furniture, assembling components, or cleaning expansive surfaces, individual manipulators often reach their physical and functional limits. Multi-degree-of-freedom (DoF) systems, such as bimanual robots and mobile manipulators, provide a scalable way to address such limitations by enabling spatial coordination and collaborative manipulation [2].

One core capability that these systems unlock is collaborative transport, where multiple agents work together to move objects that are too large, heavy, or geometrically constrained for a single robot to handle. However, this capability comes at the cost of dramatically increased complexity: planning motion trajectories for such systems requires satisfying a wide range of spatial and physical constraints,e.g., maintaining stable grasps, ensuring collision-free motion among robots and with the environment, and respecting the kinematic limits of each agent. This results in high-dimensional problems with complex nonlinear constraints in configuration and task spaces [3].

Traditional constrained optimization methods and sampling techniques such as Markov Chain Monte Carlo (MCMC) [4] offer principled approaches to constraint satisfaction in such settings.

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

However, their reliance on local gradient information often limits their capacity to explore multi-modal distributions or reason globally about the scene geometry. Recent generative approaches have shown promise; diffusion models were applied to $\mathrm{SE}(3)$ grasp synthesis [5] and to compositional object placement [6], but they typically assume fixed constraint structures and manually specified weighting schemes. Moreover, they often struggle to scale to systems with high-dimensional, inter-dependent constraints, such as those in multi-arm or mobile manipulation platforms.

To address these limitations, we introduce **Adaptive Diffusion Constrained Sampling (ADCS)**, a generative framework for high-dimensional, constraint-aware sampling in multi-DoF robotic systems. ADCS is designed to integrate both equality and inequality constraints in $\mathrm{SE}(3)$ task space while remaining adaptable to different robot morphologies and task conditions. The key novelty lies in two adaptive mechanisms. First, we introduce an *Adaptive Constrained Conditioning*, in which the model during training learns to generate feasible $\mathrm{SE}(3)$ poses conditioned on the scene geometry via Signed Distance Fields (SDFs). During sampling, we apply differentiable constraints in joint space using a chain rule, enabling flexible post-hoc adaptation to robot-specific constraints such as joint limits, reachability, and grasp feasibility. Second, we propose a *Compositional Weighting Transformer* method, in which, instead of manually specifying energy-function weights, we incorporate a Transformer-based architecture that learns to dynamically compose task-specific energy terms, allowing the system to prioritize constraints adaptively across varying tasks and contexts, to allow for optimizing challenging tasks with varied co-occurring constraints. Moreover, the CWT can handle new types of constraints without requiring retraining.

In addition, ADCS uses a sequential sampling strategy that combines Langevin dynamics, density-aware re-weighting, and Gauss-Newton refinement, which leads to faster convergence and better precision in constrained sampling. We evaluate ADCS in several collaborative manipulation scenarios, including object transport and surface pattern stippling, using both simulated and real-world multi-arm systems - a two Franka Emika Panda arm system and a bimanual TIAGo robot. Across these settings, ADCS consistently outperforms baseline approaches in terms of task success, sampling efficiency, generalization to novel scenes, and constraint satisfaction.

To sum up, our main contributions are as follows. **(i)** We propose *ADCS*, a diffusion-based framework for constraint-aware generative sampling in multi-DoF robotic systems. **(ii)** We introduce CWT, a Transformer-based mechanism that dynamically composes constraint energies, eliminating manual weight tuning and generalizing to new constraints. **(iii)** We formulate a two-stage batch-wise constrained sampling process that enables the application of task- and robot-specific constraints in joint space, even post training.

2 Preliminaries

Here, we formally define the constrained sampling problem, describe the types of constraints considered, and outline the NLP-based sampling method used to generate training data for our model.

Problem Statement. We consider a multi-DoF robotic system composed of multiple manipulators operating in a shared workspace. Let $\mathcal{Q} \subset \mathbb{R}^{n \times d}$ denote the joint configuration space, where n denotes the number of robots/end-effectors and d denotes the Dof of each robot, and let $\mathcal{C} = \{c_i\}$ be a set of task-specific constraints on configurations $q \in \mathcal{Q}$. Each constraint is either an equality $c_i(q) = 0$ or an inequality $c_i(q) \leq 0$. These constraints capture spatial relationships, kinematic feasibility, and task semantics across the robot and the environment.

Constraint Types. We consider the following types of constraints: **(i) SE(3) Pose Constraints:** enforce absolute or relative poses between end-effectors. Given two poses $H_1, H_2 \in \mathrm{SE}(3)$, the constraint is $H_1^{-1} H_2 = T_{\text{target}}$; **(ii) Orientation Constraints:** constrain either a specific axis (e.g., Y-axis of the gripper aligned with Z-axis of the world) or the full rotation $R = R_{\text{goal}}$; **(iii) Midpoint Constraints:** enforce the midpoint between two end-effectors to match a target (equality) or lie on a surface (inequality); **(iv) Signed Distance Constraints:** encode proximity to object surfaces

for surface contact or collision avoidance; **(v) Joint and Self-Collision Constraints:** enforce joint limits and prevent inter-link collisions using a set of differentiable bounding spheres.

NLP Sampling. To generate training data, we use *NLP Sampling* [4], a method for sampling from the constraint-satisfying region defined by $\mathcal{C}$. Each constraint $c_i(q)$ is mapped to a slack term: inequality constraints use $[c_i(q)]_+$, and equality constraints use $|c_i(q)|$, where $[\cdot]_+$ denotes the ReLU function. The aggregated slack vector $s(q)$ defines the relaxed energy:

$$F_{\gamma,\mu}(q) = \gamma f(q) + \mu \|s(q)\|^2, \tag{1}$$

where $f(q)$ is a task-specific density and γ, μ balance task likelihood and constraint satisfaction. Sampling proceeds in two stages: first, Gauss-Newton descent minimizes $\|s(q)\|^2$ over K_{down} steps to find a feasible configuration; then, interior sampling, such as manifold-RRT [7] or Langevin dynamics [8], is applied for K_{burn} iterations to generate diverse samples from the feasible region. This approach produces high-quality training data for constraint-aware generative modeling.

3 Adaptive Diffusion Constrained Sampling

In this section, we describe our framework for generating diverse, constraint-satisfying solutions in multi-DoF robotic systems. We first introduce our compositional energy-based formulation for modeling constraints in a diffusion framework. We then describe the model architecture, training loss, and how the learned model can be used for efficient sampling at inference time.

3.1 Adaptive Compositional Diffusion with Conditioning

Given a set of constraints $\mathcal{C} = \{c_i\}$, we aim to sample robot poses $H \in \mathrm{SE}(3)^n$ (for n end-effectors) that satisfy all constraints in $\mathcal{C}$. We model the conditional distribution over poses using an energy-based model (EBM), enabled by the use of diffusion models. Unlike standard DDPMs [8] that generate samples from noise via learned denoisers, we use diffusion training solely to learn a score function for constraint-driven energy modeling based on denoising score matching [9]. For each individual constraint $c \in \mathcal{C}$, we define an energy function $E_\theta(H \mid c)$, and the associated probability is given by $p(H \mid c) \propto \exp\big(-E_\theta(H \mid c)\big)$. We assume these models assign approximately uniform mass over the feasible region of each constraint. To satisfy all constraints jointly, we construct a composed distribution by minimizing the sum of individual energy functions

$$H_0 = \arg\min_H \sum_{c \in \mathcal{C}} E_\theta(H \mid c). \tag{2}$$

Similarly to Liu et al. [10], the joint diffusion distribution over noisy latent variables H_k (at timestep k) under all constraints $c_0, \ldots, c_N \in \mathcal{C}$ is given by

$$p(H_k \mid c_0, \ldots, c_n) \propto p(H_k) \prod_{i=0}^{N} \frac{p(H_k \mid c_i)}{p(H_k)}, \tag{3}$$

which leads to the following form for the composed energy

$$E_\theta(H_k \mid c_0, \ldots, c_n) = w_0 E_\theta(H_k \mid c_0) + \sum_{i=1}^{N} w_i \left(E_\theta(H_k \mid c_0, c_i) - E_\theta(H_k \mid c_0)\right), \tag{4}$$

where c_0 is a fundamental constraint (e.g., task-space goal, which should be satisfied in every task), and the remaining energies are conditioned on the satisfaction of c_0, since in all of our tasks, the end-effector's relative pose constraints c_0 should be satisfied. The weights w_i determine the influence of each constraint during composition and can be either fixed or learned (see Sec. 3.2).

To enable learning in the Lie group $\mathrm{SE}(3)^n$, we inject Gaussian noise in the Lie algebra $\mathfrak{se}(3)^n$, to maintain manifold consistency. Specifically, for a clean pose H, we apply a perturbation via the exponential map [5]:

$$\widehat{H} = H \cdot \mathrm{Expmap}(\epsilon), \quad \epsilon \sim \mathcal{N}(0, \sigma_k^2 I), \quad \epsilon \in \mathbb{R}^{6n}, \tag{5}$$

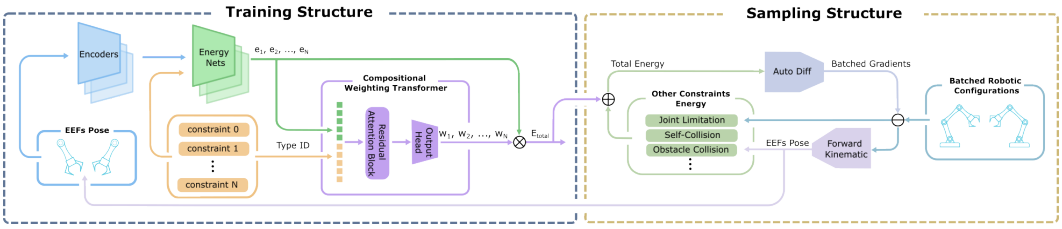


Figure 1: Overview of the Adaptive Diffusion Constrained Sampling (ADCS) architecture. The model consists of three components: a constraint feature encoder, an energy network (MLP), and the Compositional Weighting Transformer (CWT). During training, noisy poses in $\mathrm{SE}(3)^n$ are used to learn constraint-aware energy functions. In inference, sampling is performed in batched joint space using Langevin dynamics guided by the learned energy landscape.

where σ_k is the noise scale at timestep k, and Expmap maps noise vectors in the tangent space to elements of the Lie group. To train the model, we adopt a denoising score matching objective [9], which minimizes the discrepancy between the model's gradient and the gradient of the Gaussian perturbation distribution. The training loss is

$$\mathcal{L} = \frac{1}{L}\sum_{k=0}^{L} \mathbb{E}_{H,\widehat{H}} \left[\left\| \nabla_{\widehat{H}} E_\theta(\widehat{H} \mid \mathcal{C}, k) - \nabla_{\widehat{H}} \log q(\widehat{H} \mid H, \sigma_k^2 I) \right\|^2 \right], \tag{6}$$

where $q(\widehat{H} \mid H, \sigma_k^2 I)$ is the Gaussian perturbation distribution in the Lie algebra $\mathfrak{se}(3)^n$, and $\nabla_{\widehat{H}}$ denotes the gradient w.r.t. the perturbed pose $\widehat{H} \in \mathrm{SE}(3)^n$. Since $\mathrm{SE}(3)^n$ is a Lie group, gradients are computed in the associated tangent space via a local reparameterization (details in the Appendix). This formulation learns an EBM implicitly through a diffusion process. While standard diffusion models learn a generative process via denoising predictions, our framework instead learns the score function, i.e., the gradient of the log-probability—directly. The trained energy function $E_\theta(\widehat{H} \mid \mathcal{C}, k)$ thus estimates the gradient of the log-density at different noise levels. At inference time, we discard the forward diffusion and use the learned score function as input to Langevin dynamics, iteratively refining a sample by ascending the composed energy landscape as we describe in Sec. 3.3

3.2 Compositional Weighting Transformer

To enable flexible and context-aware integration of multiple constraint energies, we introduce the *Compositional Weighting Transformer (CWT)*. Rather than statically assigning scalar weights to each constraint model, the CWT learns dynamic composition weights conditioned on the current energy values and their interrelations. The key intuition is that the Transformer [11] enables *contextual weighting*: it attends to all constraint-specific energy values jointly and can adapt the relative importance of each constraint depending on their magnitudes, types, or combinations. This makes the composition mechanism *permutation-invariant* (no fixed order of constraints is required) and *adaptive to task conditions*, as the model can, for instance, downweight a nearly satisfied constraint and prioritize violated ones during inference.

As illustrated in Figure 1, each energy scalar $E_\theta(\widehat{H} \mid c_i)$ corresponding to constraint $c_i \in \mathcal{C}$ is treated as a token. These tokens are concatenated and passed through a Transformer, which outputs a set of normalized weights $\{w_i\}$. The final composed energy is $E_{\text{total}}(\widehat{H}) = \sum_i w_i E_\theta(\widehat{H} \mid c_i)$. We use positional encodings and energy's encodings for the input tokens, but unlike the original Transformer [11], in CWT, those are randomly sampled at each iteration since there is no intrinsic ordering among constraints. Since each type of constraint has its own independent energy encoder, all constraint types are implicitly encoded. This enables modularity and compositional generalization across tasks and scenes without retraining. Compared to approaches that directly merge constraint features using fully connected layers [12, 13], CWT preserves the identity of each constraint's energy and defers fusion to the composition stage. This enables each constraint energy to preserve both geometric structure and semantics, while allowing adaptive trade-offs between them as the constraint energies change with the environment. CWT is trained jointly with the energy

networks and feature encoders using the diffusion-based loss in Eq. 6, without requiring additional supervision. A full training pipeline is provided in A.2 and Algorithm 1 in the Appendix.

3.3 Two-stage Batch-wise Constrained Sampling

At inference time, we use the learned diffusion model to generate batched joint configurations that satisfy a set of geometric constraints. To this end, we adopt a sequential sampling strategy inspired by Toussaint et al. [4], which combines Annealed Langevin Monte Carlo (ALMC) sampling with resampling and correction steps. The procedure is summarized in Algorithm 2 in the Appendix.

Annealed Langevin Sampling. In each ALMC call, we first draw a batch of joint configurations $q_L^{n_s}$ from a standard Gaussian distribution. Using PyTorch Kinematics [14], we compute the corresponding end-effector poses $H_L^{n_s} \in \mathrm{SE}(3)^n$ via forward kinematics. These poses are then passed through the energy network to compute their corresponding energies $E_\theta(H_L^{n_s} \mid \mathcal{C}, k)$. Next, we compute the gradient of the energy w.r.t. the joint configuration $q_k^{n_s}$ using PyTorch's AutoDiff [15]. This gradient is obtained via the chain rule $\nabla_q E_\theta = \frac{\partial E_\theta(H_k|\mathcal{C},k)}{\partial H_k} \cdot \frac{\partial H_k}{\partial q_k^{n_s}}$. The joint configurations are then updated using a Langevin step. To improve convergence, we adopt a second-order strategy similar to Carvalho et al. [16], but instead of relying solely on first-order gradients, we introduce an approximate batch-wise Gauss-Newton update during this correction phase. The update direction is computed as $g_{n_s} = -(J^\top J)^{-1} J^\top r$, where $J = \frac{\partial r}{\partial q} = \frac{\partial r}{\partial H}\frac{\partial H}{\partial q}$ and r is the residual value. This approximation accelerates convergence while preserving the structure of the learned data distribution.

Following each ALMC sampling pass, we perform a deterministic *second-order refinement* step to improve the quality and constraint satisfaction of the sampled configurations. In this phase, we reduce the step size and omit the Gaussian noise, allowing the model to converge more precisely. We apply a Gauss-Newton correction g_{n_s} during this step, using approximate second-order information to guide updates in joint space. Importantly, we restrict this correction to the refinement phase to avoid distorting the learned distribution during stochastic sampling, as discussed in [17, 18].

Resampling Phase. After the initial ALMC sampling and second-order refinement, we perform a resampling step to promote diversity among the generated configurations. We begin by sorting all samples based on their energy values and selecting a fixed number of the top-performing configurations. To avoid over-representing densely clustered solutions, we estimate the sample density $\rho(x)$ using Kernel Density Estimation (KDE) with a Gaussian kernel [19]:

$$\rho(x) = \frac{1}{M\,h} \sum_{i=1}^{M} K\left(\frac{x - x_i}{h}\right), \quad K(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right), \tag{7}$$

where M is the number of retained samples, x_i are the sampled configurations, and h is the kernel bandwidth. Replication weights are assigned inversely proportional to $\rho(x_i)$: samples in dense regions are duplicated less frequently, while those in sparse regions are upweighted. This reweighting encourages broader exploration of the configuration space, improving the final sample diversity.

4 Experimental Results

We evaluate our method, **ADCS**, across a suite of constrained motion generation tasks and compare it to representative baselines: ***Gauss–Newton*** uses the predefined sampling cost, with updates computed as $\Delta q = -(J^\top J)^{-1} J^\top r$. ***Diffusion-CCSP*** [6] follows a similar energy-based formulation, with fixed (uniform) constraint weights and annealed Langevin dynamics as the sampler. ***NLP-Sampling*** uses the method from [4], note that its internal optimization loop is not externally controllable, so we specify only the number of final samples. All methods are evaluated over 10 random seeds (0–9), and we report average metrics across these runs. For fair comparison, Gauss–Newton and CCSP are run for 600 iterations, matching the sampling budget of ADCS.

4.1 Simulation Experiment

We design four simulation tasks featuring increasingly complex geometric and pose constraints (see Figure 2). All tasks are performed using a dual-arm Franka robot. For each task, we generate 500 samples using each method (batch size is 500).

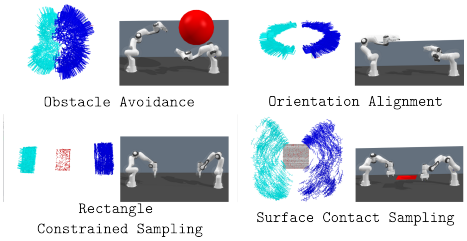


Figure 2: Sampled configurations for simulation tasks.

	Obstacle Avoidance	Orientation Alignment	Rectangle Constrained Sampling	Surface Contact Sampling
Data Distribution Coverage				
NLP Sampling	**0.0321**	0.0235	0.0425	-
CCSP	0.0310	0.0342	0.41	0.2407
ADCS (ours)	0.0315	**0.0362**	**0.6**	**0.6425**
Data Distribution Uniformity (density variance)				
NLP Sampling	0.3388	0.5260	126.7000	-
CCSP	**0.1687**	0.1548	13.7969	5.0834
ADCS (ours)	0.3669	**0.1338**	**1.9097**	**1.5696**

Table 1: Comparison of **coverage as sampled** and **data uniformity** by NLP Sampling, CCSP and ADCS.

All tasks involve a relative pose constraint between the two end-effectors and a midpoint constraint. **(1)** *Obstacle Avoidance*: relative pose + symmetric midpoint constraint and adds a spherical obstacle to avoid. **(2)** *Orientation Alignment:* adds orientation alignment with the table plane without obstacle, i.e., the EEF's Y-axis is aligned with the world's Z-axis. **(3)** *Rectangle Constrained Sampling:* midpoint lies inside a box (defined via bounds), with fixed EEF rotation matrix. **(4)** *Surface Contact Sampling:* replaces the region with a surface defined by a point cloud and loosens the orientation constraint to be normal to the surface, i.e., the EEF's Z-axis is aligned with the negative world Z-axis.

In Table 2, we report **mean, median, and third-quartile values for position and rotation errors** across four tasks. Under a fixed sampling budget (600 iterations), ADCS consistently yields significantly lower constraint satisfaction errors than Gauss–Newton and Diffusion-CCSP. Especially in tasks with smaller feasible regions, such as Rectangle Constrained Sampling, our method demonstrates stronger performance compared to the baselines. And compared to the state-of-the-art NLP-Sampling, it outperforms in most metrics while requiring shorter sampling time. In contrast, ADCS offers strong accuracy with significantly improved efficiency and broader applicability.

We also compare the **spatial quality of samples** generated by NLP Sampling, CCSP, and ADCS in Table 1, evaluating both **coverage** and **uniformity** of the resulting distributions. Task *Obstacle Avoidance* and *Orientation Alignment* focus on the 3D position of a single end-effector, while Task *Rectangle Constrained Sampling* and *Surface Contact Sampling* assess the midpoint distribution between two arms. Coverage is computed as the fraction of occupied voxels in a predefined 3D grid, while uniformity is measured by the variance in density across voxels, lower variance indicates a more even spread of samples. As shown in the table, ADCS achieves superior coverage in most tasks and lower variance in most cases, highlighting its ability to explore constraint-satisfying regions more thoroughly.

4.2 Real-World Experiment

We have designed eight different tasks that integrate our ADCS with motion planning. Under various specified constraints, these eight tasks involve stippling operations, i.e., two Franka robots collaboratively grasp a pen to perform different stippling tasks as shown in Figure 3. In addition, we conducted tests on the TIAGo robot, allowing it to carry objects using both hands.

To integrate with motion planning, we first perform radius-based thinning on the original sampled points to filter out all points within a specified radius of the selected points. We then sort the remaining points using different, task-specific sampling and sorting strategies. After sorting, we inspect the distance between consecutive points in the joint space. Whenever the distance exceeds a preset threshold, we use the Gauss-Newton [20] method to project to obtain a closer joint configuration, which usually converges in just two to three iterations.

	RPE (mm)		MPE (mm)		RRE (rad)		ERE (rad)		Time (s)
	Median	Q3	Median	Q3	Median	Q3	Median	Q3	Mean
Obstacle Avoidance									
NLP Sampling	0.0180	0.0660	0.0070	0.0292	4.652e-5	0.0001	-	-	61.4200
Gauss-Newton	**0.0001**	0.0003	7.649e-5	0.0001	1.718e-7	3.215e-7	-	-	**6.5329**
CCSP	5.7985	23.6867	9.9820	31.7004	0.0128	0.0373	-	-	9.1025
ADCS (ours)	**0.0001**	**0.0002**	**6.837e-5**	**8.652e-5**	**1.560e-7**	**2.149e-7**	-	-	10.3321
Orientation Alignment									
NLP Sampling	0.0358	0.0639	0.0177	0.0350	7.012e-5	0.0001	0.0002	0.0003	72.8657
Gauss-Newton	13.5566	109.0678	2.1668	46.6404	0.0366	0.3812	3.576e-07	0.0185	**7.6832**
CCSP	12.5211	38.7049	11.1668	38.5522	0.0101	0.0309	0.0341	0.0671	9.3460
ADCS (ours)	**0.0050**	**0.0060**	**0.0026**	**0.0029**	**1.495e-5**	**1.611e-5**	**1.192e-7**	**1.788e-7**	10.0325
Rectangle Constrained Sampling									
NLP Sampling	0.0176	0.0712	1.0019	4.4726	0.00003	0.00009	0.00001	0.00004	61.3578
Gauss-Newton	0.0008	121.098	0.3174	9.4909	1.412e-06	0.2704	5.277e-7	3.418e-5	**7.7406**
CCSP	170.4515	257.2366	21.8248	56.5533	0.0458	0.1007	0.1334	0.2218	10.3574
ADCS (ours)	**0.0001**	**0.0002**	**0.1690**	**0.2267**	**1.727e-7**	**2.394e-7**	**1.334e-7**	**1.883e-7**	11.3247
Surface Contact Sampling									
NLP Sampling	-	-	-	-	-	-	-	-	-
Gauss-Newton	66.8540	247.1499	4.4389	44.8222	0.0875	0.5902	0.0003	0.0188	**8.0048**
CCSP	23.7275	58.6257	11.3875	23.6884	0.0282	0.0691	0.0710	0.1411	13.7951
ADCS (ours)	**0.0010**	**0.0013**	**1.8522**	**2.8253**	**7.439e-7**	**9.614e-7**	**0**	**5.960e-8**	14.9514

Table 2: **Comparison of Gauss–Newton, CCSP, and ADCS** across 4 tasks. For each task and method, we report **RPE** (relative position error (mm)), **MPE** (mid-point position error (mm)), **RRE** (relative rotation error (rad)), and **ERE** (end-effector rotation error in the world frame (rad)) by median and third quartile (Q3), and average computation time (s).

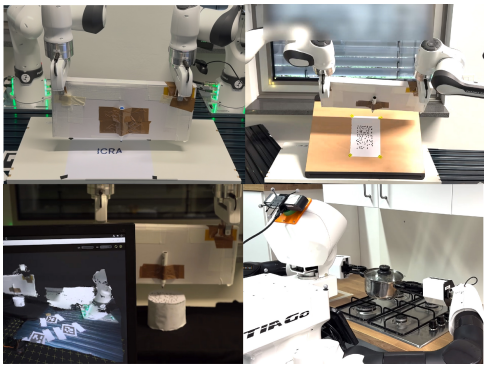


Figure 3: The real-world experimental setup involves two Franka robots collaboratively grasping a pen for stippling operations and a TIAGo to carry objects using both hands.

	RPE	MPE	RRE	ERE	VPR
Fixed Midpoint with Orientation Alignment	6.9647	1.6872	0.0074	0.0042	72.60
Circular ConstrainedStippling	0.0706	0.0024	0.0001	4.27e-05	98.40
Inclined Circular Constrained Stippling	0.2818	0.0018	0.0003	0.0001	88.40
Rectangle Constrained Stippling	0.1541	0.0097	0.0002	0.0002	90.60
Inclined Rectangle Constrained Stippling	0.6734	0.0312	0.0006	0.0007	84.80
Letter Pattern Stippling	2.7895	0.0316	0.0085	0.0101	80.40
Cube Surface Stippling	2.0677	0.0387	0.0055	0.0063	82.45
Cylinder Surface Stippling	2.3240	0.0282	0.0058	0.0087	80.70

Table 3: The **constraint errors (Q3)** and the **ratio of valid points (VPR (%))** of different tasks in the real-world.

As illustrated in Figure 4, we evaluate eight real-world tasks. All tasks involve a relative pose constraint between the two end-effectors, as well as orientation constraints for the end-effectors. **(1)** *Fixed Midpoint with Orientation Alignment*: constrains the midpoint between the two arms to a fixed location. **(2)** *Circular Constrained Stippling* and **(3)** *Inclined Circular Constrained Stippling*: restrict the midpoint to lie within a (inclined) circular bounding box. **(4)** *Rectangle Constrained Stippling* and **(5)** *Inclined Rectangle Constrained Stippling*: restrict the midpoint to lie within a (inclined) rectangle bounding box. **(6)** *Letter Pattern Stippling* requires the midpoint to follow point clouds generated from mesh models of letters. Finally, **(7)** *Cube Surface Stippling* and **(8)** *Cylinder Surface Stippling*: constrain the midpoint to the top surface of real objects, with point clouds obtained directly from a camera.

In Table 3, we present the constraint error and the ratio of valid points for each task. Note that each task was tested ten times, and in every task, 500 points were generated through sampling. We consider a point valid if its positional error is less than 3 mm and its orientational error is less than 0.005 rad. As shown, the probability of valid points exceeds 80% in most tasks.

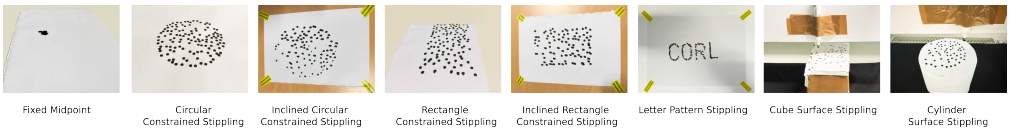


Figure 4: Final stippling layout of 8 different real-world tasks.

4.3 Generalization and Robustness Capability

To evaluate the generalization capability of our model, we tested the constraint values outside of the training data under different tasks. The detailed results are provided in the Appendix A.9.

5 Related Works

Constrained Sampling. Sampling under constraints is central to problems in MCMC, optimization, and robotics. With equality constraints, the task reduces to sampling on a manifold [7, 21], while linear inequalities lead to polytope sampling [22]. Toussaint et al. [4] proposed a restart-based NLP sampler for nonlinear constraints in robotics, assuming only local access to cost and constraint functions. In contrast, we focus on sampling from complex, multimodal constrained distributions.
Generative Models for Constrained Optimization. Deep generative models such as DDPM [8] and SVGD [23] are powerful tools for sampling from multimodal distributions. Their ability to capture disconnected regions makes them attractive for constraint-aware inference [24]. Existing works [25, 6] apply diffusion models to constraint-aware tasks, but often focus on low-dimensional object poses or categorical approximations. Diffusion-based motion planners [16, 26, 27] reframe planning as sampling in energy-based models, but struggle to scale with multiple, nonlinear constraints. Our method addresses this by learning a compositional energy model with transformer-based adaptive weighting for rich constraint integration.
Automatic Weight Tuning. Designing appropriate constraint weights is tedious and sensitive [5, 28]. Prior methods either optimize weights [29, 30] or learn them from data [31, 32], but these are not directly applicable to diffusion-based models. Recent approaches [5, 6, 16] use fixed weights, whereas our method trains a transformer (CWT) to reason over constraint types and adaptively weight them at inference.

6 Conclusion

In this paper, we introduced Adaptive Diffusion Constrained Sampling (ADCS), a generative framework designed to effectively integrate equality and inequality geometric constraints within multi-DoF robot manipulation tasks. Our approach uses a dedicated energy network to calculate the cost of equality and inequality constraints, and introduces an SDF network to incorporate external environment perception, while combined with a Transformer-based architecture for dynamically weighting these constraints during inference. Experimental evaluations across four simulation tasks and eight real-world tasks demonstrated that ADCS outperforms baseline methods, also applicable to scenarios with small feasible areas and strict spatial constraints. Additionally, through ablation studies, we verified that transformer-based dynamic weights, KDE-based resampling, and batch-wise constrained sampling significantly enhance sample uniformity and performance. Moreover, our sampler can be used to construct a roadmap for enabling motion planning. Finally, our method's robust generalization was validated on out-of-distribution data, confirming its adaptability and efficiency in realistic robotic applications.

7 Limitation

In this work, the SE(3) constraints are defined by manually specifying corresponding numerical values, rather than through instructions like "align both hands". An interesting direction would be to combine this with Vision-Language Models (VLMs) and define the related constraints through Large Language Models (LLMs). Furthermore, LLMs could be employed to define cost functions for the Gauss-Newton refinement, thereby allowing a more general optimization framework. Lastly, the current training process does not incorporate information about the corresponding joint configurations. Future improvements include joint information during the training phase to improve model performance and accelerate convergence during inference.

Acknowledgments

This project has received funding from the European Union's Horizon Europe programme under Grant Agreement No. 101120823, project MANiBOT.

References

[1] J. Canny. *The complexity of robot motion planning*. MIT press, 1988.

[2] O. Khatib. Mobile manipulation: The robotic assistant. *Robotics and Autonomous Systems*, 26 (2-3):175–183, 1999.

[3] Z. Kingston, M. Moll, and L. E. Kavraki. Sampling-based methods for motion planning with constraints. *Annual review of control, robotics, and autonomous systems*, 1(1):159–185, 2018.

[4] M. Toussaint, C. V. Braun, and J. Ortiz-Haro. Nlp sampling: Combining mcmc and nlp methods for diverse constrained sampling. *arXiv preprint arXiv:2407.03035*, 2024.

[5] J. Urain, N. Funk, J. Peters, and G. Chalvatzaki. Se (3)-diffusionfields: Learning smooth cost functions for joint grasp and motion optimization through diffusion. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5923–5930. IEEE, 2023.

[6] Z. Yang, J. Mao, Y. Du, J. Wu, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Compositional diffusion-based continuous constraint solvers. *arXiv preprint arXiv:2309.00966*, 2023.

[7] C. Suh, T. T. Um, B. Kim, H. Noh, M. Kim, and F. C. Park. Tangent space rrt: A randomized planning algorithm on constraint manifolds. In *2011 IEEE International Conference on Robotics and Automation*, pages 4968–4973. IEEE, 2011.

[8] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[9] Y. Song and S. Ermon. Improved techniques for training score-based generative models. *Advances in neural information processing systems*, 33:12438–12448, 2020.

[10] N. Liu, S. Li, Y. Du, A. Torralba, and J. B. Tenenbaum. Compositional visual generation with composable diffusion models. In *European Conference on Computer Vision*, pages 423–439. Springer, 2022.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[12] D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[13] X. Jia, B. De Brabandere, T. Tuytelaars, and L. V. Gool. Dynamic filter networks. *Advances in neural information processing systems*, 29, 2016.

[14] S. Zhong, T. Power, A. Gupta, and P. Mitrano. PyTorch Kinematics, 2024.

[15] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[16] 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 *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1916–1923. IEEE, 2023.

[17] M. Girolami and B. Calderhead. Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 73(2): 123–214, 2011.

[18] B. Leimkuhler and C. Matthews. Rational construction of stochastic numerical methods for molecular sampling. *Applied Mathematics Research eXpress*, 2013(1):34–56, 2013.

[19] B. W. Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

[20] C. F. Gauss and C. H. Davis. Theory of the motion of the heavenly bodies moving about the sun in conic sections. *Gauss's Theoria Motus*, 76(1):5–23, 1857.

[21] B. Kim, T. T. Um, C. Suh, and F. C. Park. Tangent bundle rrt: A randomized algorithm for constrained motion planning. *Robotica*, 34(1):202–225, 2016.

[22] Y. Chen, R. Dwivedi, M. J. Wainwright, and B. Yu. Fast mcmc sampling algorithms on polytopes. *Journal of Machine Learning Research*, 19(55):1–86, 2018.

[23] Q. Liu and D. Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. *Advances in neural information processing systems*, 29, 2016.

[24] J. Urain, A. Mandlekar, Y. Du, M. Shafiullah, D. Xu, K. Fragkiadaki, G. Chalvatzaki, and J. Peters. Deep generative models in robotics: A survey on learning from multimodal demonstrations. *arXiv preprint arXiv:2408.04380*, 2024.

[25] J. Ortiz-Haro, J.-S. Ha, D. Driess, and M. Toussaint. Structured deep generative models for sampling on constraint manifolds in sequential manipulation. In *Conference on Robot Learning*, pages 213–223. PMLR, 2022.

[26] V. Kurtz and J. W. Burdick. Equality constrained diffusion for direct trajectory optimization. *arXiv preprint arXiv:2410.01939*, 2024.

[27] C. Pan, Z. Yi, G. Shi, and G. Qu. Model-based diffusion for trajectory optimization. *Advances in Neural Information Processing Systems*, 37:57914–57943, 2024.

[28] Y. Zhang, C. Pezzato, E. Trevisan, C. Salmi, C. H. Corbato, and J. Alonso-Mora. Multimodal mppi and active inference for reactive task and motion planning. *IEEE Robotics and Automation Letters*, 2024.

[29] A. R. Kumar and P. J. Ramadge. Diffloop: Tuning pid controllers by differentiating through the feedback loop. In *2021 55th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6. IEEE, 2021.

[30] S. Cheng, M. Kim, L. Song, C. Yang, Y. Jin, S. Wang, and N. Hovakimyan. Difftune: Autotuning through auto-differentiation. *IEEE Transactions on Robotics*, 2024.

[31] A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe. Automatic lqr tuning based on gaussian process global optimization. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 270–277. IEEE, 2016.

[32] A. Loquercio, A. Saviolo, and D. Scaramuzza. Autotune: Controller tuning for high-speed flight. *IEEE Robotics and Automation Letters*, 7(2):4432–4439, 2022.

[33] L. Pineda, T. Fan, M. Monge, S. Venkataraman, P. Sodhi, R. T. Chen, J. Ortiz, D. DeTone, A. Wang, S. Anderson, J. Dong, B. Amos, and M. Mukadam. Theseus: A Library for Differentiable Nonlinear Optimization. *Advances in Neural Information Processing Systems*, 2022.

[34] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots. Continuous-time gaussian process motion planning via probabilistic inference. *The International Journal of Robotics Research*, 37(11):1319–1340, 2018.

[35] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019.

[36] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

A Appendix

A.1 Cost Functions for Sampling

We consider these cost functions in the tasks:

(1) **SE(3) Cost**: Given two points $H_1 = \begin{bmatrix} R_1 & t_1 \\ 0 & 1 \end{bmatrix}$, $H_2 = \begin{bmatrix} R_2 & t_2 \\ 0 & 1 \end{bmatrix} \in SE(3)$, the cost can be defined as $c_{\mathrm{SE3}} = c_t + c_r$, where $c_t = \|t_1 - t_2\|^2$, $c_r = \mathrm{Log}(R_1 \top R_2)$, while $\mathrm{Log}(\cdot)$ maps a rotation matrix $R \in \mathrm{SO}(3)$ into its tangent space Lie-algebra $\mathfrak{so}(3)$, here we use Theseus [33] to compute this.

(2)**SDF Cost**: Here we use the SDF value computing by [14] directly, i.e., $c_{\mathrm{sdf}}(H_t) = \mathrm{SDF}(H_t)$.

(3) **Obstacle Collision Cost**: Similar to the work of Mukadam et al. [34], we instantiate N *collision spheres* for the robot. By using differentiable PyTorch kinematics [14] to compute the forward kinematics, we obtain the position $p(q, s_n)$of each collision sphere, where q denotes the robot's joint configuration. We then compute the distance to the obstacle via a differentiable signed distance function $d(p, obs)$. For any joint configuration, the obstacle collision cost can be expressed as: $c_{obs}(q) = \sum_i^N (d(p_i, obs) - \epsilon_i)$, where ϵ is the radius of the collision sphere.

(4) **Self-Collision Cost**: Similar to the obstacle collision cost, we attach a set of collision spheres as a group to each link for collision detection, and then compute the distances between each pair of sphere's groups as the self-collision cost.

(5) **Joint Limit Cost**: We constrain the joints within specified ranges by incorporating an L2 norm and introduce a degree ϵ of redundancy to minimize edge cases as much as possible.

A.2 Training Structure

Training Architecture In order to make the trained model flexibly applicable to different types of robots, we perform data noising in SE(3) space during the training phase, and then map to the joint space for sampling during the inference phase. In this work, we focus primarily on SE(3) constraint problems, with each constraint energy network implemented as a five-layer multilayer perceptron (MLP). We manually define feature functions $f_c(\cdot)$ for SE(3) constraints to capture the target feature. These features are then encoded and input into the energy network. We also incorporate the object's point-cloud information as an inequality constraint, e.g., ensuring that sampled points lie within a specified region. To enable our model to generalize better across different object shapes, we define a learnable shape code z_o as in work [35], and jointly train a network to predict SDF values with given z_o, replacing a hand-defined feature function, so that model can closely approximate the true SDF values. Here the ground-truth SDF values are computed during training using [14]. Finally, the predicted SDF values together with the shape code z_o are fed as inputs into the energy network.

As shown in Alg. 1, our training process consists of four main parts. **First**, we use the point-cloud encoder F_θ^{pc} to map the raw point cloud o to a latent code z_o, then feed z_o and the perturbed data $\hat{H}$ into the SDF network F_θ^{sdf} to predict the SDF values $\hat{s}$, and compute the prediction loss $L_{\mathrm{sdf}}(\hat{s}, s)$ against the ground-truth SDF s. **Second**, for each predefined geometric or physical constraint c_i, we extract the corresponding feature f_i from $\hat{H}$ and estimate its energy $e_i = E_\theta(f_i)$. If a constraint is inactive, we simply set $e_i = 0$. **Third**, we concatenate the energies $\{e_i\}$ as tokens and feed them into the Transformer $F_\theta^{\mathrm{trans}}$, which produces a set of weights $\{w_i\}$. The total energy is then computed as $E = \sum_i e_i\, w_i$. **Finally**, we get the L_{diff} from Eq. 6 and combine the two losses, while back-propagate to update all model parameters θ, thereby improving both SDF accuracy and constraint consistency.

It is worth noting that our model is jointly trained. However, instead of training on all constraints simultaneously in every iteration, we randomly sample a subset or a single constraint for each training step. This strategy allows the model to better capture the relationships and interactions among different constraints.

Algorithm 1: Adaptive Diffusion Constraint Model Training

Given : θ_0: initial params for F_θ^{sdf},F_θ^{pc},$F_\theta^{\text{feature}}$,$F_\theta^{\text{trans}}$, E_θ,
C : constraints set,
N: maximum number of constraints

Datasets: $\mathcal{D}_o : \{o\}$, object's point cloud,
$\mathcal{D}_H : \{H\}$, training data $H \in \text{SE(3)}$;

for $s \leftarrow 0$ **to** $S-1$ **do**
  Sample noise scale $k, \sigma_k \leftarrow [0, \ldots, L]$;
  Sample object point cloud $o \in \mathcal{D}_o$;
  Sample $H \sim \mathcal{D}_H$;
  $\epsilon \sim \mathcal{N}(0, \sigma_k^2 I)$;
  $\hat{H} = H\text{ExpMap}(\epsilon)$; // Add the noise
  SDF Train
  $z_o \leftarrow F_\theta^{\text{pc}}(o)$; // Encode the point cloud
  $\hat{s} \leftarrow F_\theta^{\text{sdf}}(z_o, \hat{H})$; // Predcit the SDF value
  $s \leftarrow \text{SDF}(\hat{H})$; // Compute the ground-truth SDF value
  $L_{\text{sdf}} \leftarrow \mathcal{L}(\hat{s}, s)$;
  Energy Net Train
  for $i = 0$ **to** $N-1$ **do**
    if $c_i \in C$ **then**
      $f_i \leftarrow F_\theta^{\text{feature}}(\hat{H}, c_i)$; // Get the features from each constraint
      $e_i \leftarrow E_\theta(f_i; \theta)$; // Estimate each energy
    else
      $e_i \leftarrow 0$; // Zero padding
    end
  end
  Transfomer-based Weights Train
  $\{w_i\}_{i=0}^{N-1} \leftarrow F_\theta^{\text{trans}}(\{e_i\}_{i=0}^{N-1})$; // Compute the weights using Transformer
  $e_{total} \leftarrow \sum_{i=0}^{N-1} e_i \cdot w_i$;
  $L_{\text{diff}} = \mathcal{L}(e_{total}, \hat{H}, H, \sigma_k)$; // Compute the loss ▷ Eq. 6
  Parameter Update
  $L \leftarrow L_{\text{sdf}} + L_{\text{diff}}$;
  $\theta \leftarrow \theta - \eta \nabla_\theta L$; // Update the parameters θ
end

A.3 Sampling Structure

Algorithm 2: Adaptive Diffusion Constraint Sampler

Input: $\{\sigma_k\}_{k=1}^{L}$: Noise levels;
L: Diffusion steps;
ϵ: Step rate;
n_s: Number of initial samples;
n_r: Number of resampling samples;
t_f: Fixed time step;
α_f: Fixed step rate;
C: Constraint set
Output: Final samples $q_0^{n_s}$

Initialize $q_L^{n_s} \sim p_L(q)$;
$q_0^{n_s}, e_0^{n_s} \leftarrow \text{ALMC}(q_L^{n_s}, \sigma_k, L, \epsilon, t_f, \alpha_f, C)$; // Get the joint configurations and energy after fisrt stage sampling ▷ Alg. 3
$q_0^{n_r} \leftarrow \text{sort}(q_0^{n_s}, e_0^{n_s})$; // Select data with less energy
$\rho(q) \leftarrow \text{KDE}(q_0^{n_r})$; // Estimate density
$w_i \propto 1/\rho(q_i)$; // Compute the replication weights
$q_L^{n_s} \leftarrow \text{repeat}(q_0^{n_r}, w_i)$; // Repeat data based on weights
$q_0^{n_s}, e_0^{n_s} \leftarrow \text{ALMC}(q_L^{n_s}, \sigma_k, L, \epsilon, t_f, \alpha_f, C)$; // Get final $q_0^{n_s}$ ▷ Alg. 3

Algorithm 3: Annealed Langevin Markov Chain Monte Carlo Sampler (ALMC)

Input: $\{\sigma_k\}_{k=1}^{L}$, L, ϵ, n_s, t_f, α_f, $q_L^{n_s}$, C
Output: Final samples $q_0^{n_s}$

for $k \leftarrow L$ **to** 1 **do**
  $H_k^{n_s} \leftarrow \text{FK}(q_k^{n_s})$;
  $e_{n_s} \leftarrow E_\theta(H_k^{n_s}, k, C)$; // Compute the energy
  $\alpha_k \leftarrow \epsilon \cdot \sigma_k / \sigma_L$; // Select step size
  $\epsilon \sim \mathcal{N}(0, I)$; // Sample white noise in joint space
  $q_{k-1}^{n_s} \leftarrow q_k^{n_s} + \left(-\frac{\alpha_k^2}{2} \frac{\partial e_{n_s}}{\partial q_k^{n_s}} + \alpha_k \epsilon\right)$; // Make a Langevin dynamics step
end
for $k \leftarrow L$ **to** 1 **do**
  $H_k^{n_s} \leftarrow \text{FK}(q_k^{n_s})$;
  $e_{n_s} \leftarrow E_\theta(H_k^{n_s}, t_f, C)$; // Compute the energy
  $r_g \leftarrow f(H_k^{n_s}, C)$; // Compute the sampling cost
  $J_g = \frac{\partial r_g}{\partial q_k^{n_s}}$; // Compute the jacobian
  $g_{n_s} \leftarrow (J_g \top J_g)^{-1} J_g \top r_g$; // Compute the Gauss Newton
  $q_{k-1}^{n_s} \leftarrow q_k^{n_s} + \left(-\frac{\alpha_f^2}{2} \frac{\partial e_{n_s}}{\partial q_k^{n_s}} - g_{n_s}\right)$; // Make a Langevin dynamics step
end
return $q_0^{n_s}$;

Here we describe our sequential sampling structure. It consists of two stages, each of which uses the ALMC algorithm. Between these stages, we perform resampling via a KDE-based replication method. The sampler consists of two nested components:

Adaptive Diffusion Constraint Sampler: Generate n_s initial joint configurations from $\mathcal{N}(0, I)$ and run one pass of ALMC to compute their energies at each noise level. Sort these samples by energy, estimate their density via KDE, and compute replication weights to resample n_r configurations. Finally, run a second ALMC pass on the resampled set to produce the final samples.

Annealed Langevin MCMC: For each noise level σ_k, perform Langevin updates by computing $\nabla_q E$ via the chain rule $\frac{\partial E}{\partial H} \frac{\partial H}{\partial q}$ and injecting Gaussian noise. Then enter a deterministic refinement phase where noise is dropped and an approximate Gauss–Newton correction $(J^\top J)^{-1} J^\top r$ is applied to accelerate convergence. And in this work, we also adopt a batch-wise Gauss–Newton optimization scheme, where multiple constraint instances are solved simultaneously within each iter-

ation. Instead of updating the parameters using a single constraint residual, the algorithm aggregates the Jacobians and residuals from an entire batch, forming a block-structured normal equation. This design leverages parallel computation, improves numerical stability, and accelerates convergence by exploiting shared information across similar problem instances.

A.4 Comparison with Diffusion-CCSP

In our work, we consider Diffusion-CCSP as the primary baseline for comparison. Compared to Diffusion-CCSP, our approach introduces several improvements that lead to stronger performance. While we retain its original network structure, which composes multiple constraint objectives through an energy-based model, for a fair comparison, there are key differences. First, unlike Diffusion-CCSP, our method incorporates the task-specific feature functions from ADCS, which are essential for enhancing generalization. Second, we find that, in particular, Diffusion-CCSP uses fixed composing weights and relying on the ULA sampler during inference, which significantly limit its ability to adapt to different constrained problems and sample effectively. In contrast, our method benefits from transformer-based learnable constraint weighting and a more expressive sequential sampling strategy, resulting in better performance across tasks.

In summary, while CCSP introduced compositionality in diffusion for scene-level object constraints, our work extends this principle to structured, high-dimensional robot constraints. With adaptive weighting, multi-type constraint fusion, and real-world execution, our approach effectively connects abstract constraint composition with practical embodied planning.

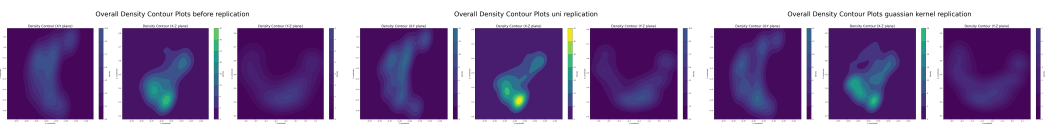
A.5 Voxel-based Statistical Method

We also employ a voxel-based statistical method [36] to compute the density of the data distribution. The principle is to uniformly divide the point-cloud's bounding box into M^3 small cubes (voxels), count the number n_i of points in each voxel, and then assess the dispersion of these counts, as Eq. 8, σ^2 represents the variance, a smaller variance indicates a more uniform data distribution. And r_c represent the coverage of the data.

$$\sigma^2 = \frac{1}{M}\sum_{i=1}^{M}\big(n_i - \bar{n}\big)^2, \quad r_c = \frac{\sum_{i=1}^{M}\mathbf{1}(n_i > 0)}{M}, \quad \mathbf{1}(n>0) = \begin{cases} 1, & n > 0, \\ 0, & n \le 0. \end{cases} \tag{8}$$

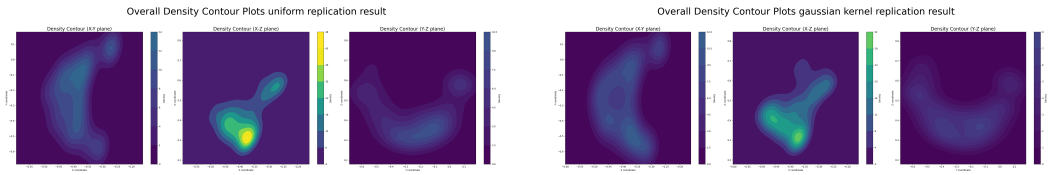
A.6 KDE-based Resampling

Under Task *Obstacle Avoidance*, we compared KDE-based resampling against uniform-replication resampling, as shown in Figure 5, the KDE-based resampling produces a more uniform distribution along each axis and exhibits fewer voids. Moreover, as shown in Figure 6, the final samples obtained through Gaussian-kernel replication are also more uniformly distributed. We estimate the data distribution density with both voxel size 5 and 10. From Table 4, we can conclude that the KDE-based resampling method achieves a more uniform data distribution both after the resampling stage and in the final sampling results.



(a) Before replication (b) Uniform replication (c) KDE-based replication

Figure 5: This figure illustrates the one end-effector's position distribution densities along the x, y, and z axes before replication, after uniform replication, and after Gaussian-kernel-based replication.



(a) Uniform replication sampling result

(b) KDE-based replication sampling result

Figure 6: This figure shows the one end-effector's position distribution densities along the x, y, and z axes of the final sampling results after applying uniform replication and after applying Gaussian-kernel–based replication.

Data Distribution Uniformity (Density Variance)				
	After Replication(5)	Final Sampled(5)	After Replication(10)	Final Sampled(10)
Uniform replication	78.0000	79.6028	5.5500	5.0039
KDE replication	**50.4320**	**57.4560**	**4.9580**	**2.8893**

Table 4: The table presents the variance of the end-effector's positional density distribution for the uniform-replication and KDE-based-replication methods, both after the resampling stage and after the final sampling. Smaller variance value indicate a more uniform distribution. We performed out these evaluations using voxel sizes of 5 and 10.

A.7 Dynamic Weighting via CWT

We compare three composition mechanisms for constraint weighting: fixed scalar weights, MLP-based dynamic weights, and our proposed Compositional Weighting Transformer (CWT). All models share the same architecture for the energy and feature networks and are trained without task likelihoods to isolate the effect of the weighting mechanism. As shown in Table 5, CWT consistently outperforms the other approaches, achieving lower errors across all tasks, underscoring its ability to adaptively balance constraint satisfaction in diverse compositions.

	RPE	MPE	RRE	ERE
Orientation Alignment				
Fixed Weights	42.5589	43.4433	0.0403	0.0660
MLP-based	40.9558	43.1051	0.0403	0.0668
CWT	**35.0300**	**34.7487**	**0.0337**	**0.0640**
Surface Contact Sampling				
Fixed Weights	63.3622	27.1177	0.0682	0.1195
MLP-based	73.4915	27.7264	0.0724	0.1014
CWT	**62.1286**	**20.2202**	**0.0393**	**0.0857**

Table 5: Comparison of constraint composition mechanisms in ADCS: fixed weights, MLP-based dynamic weights, and the proposed Compositional Weighting Transformer (CWT). Note that there is no two-stage sampling involved in this ablation and all variants use the same energy and feature networks and exclude task likelihood information during training. We report the third quartile (Q3) error for evaluation.

A.8 Two-stage Batch-wise Sampling

To evaluate the performance gains brought by two-stage sampling, we tested ADCS and ADCS without two-stage sampling on three different tasks. And each task uses the same threshold, a sample point is considered valid if its error is smaller than the threshold. As shown in Table 6, for each task, two-stage sampling consistently yields performance improvements, especially on tasks with smaller feasible regions.

Valid Sample Rate (%)			
	Obstacle Avoidance	Orientation Alignment	Reactangle Constrained Sampling
w/o Two-stage	82.3	3.74	1.54
Two-stage	**99.46**	**99.24**	**99.7**

Table 6: Comparison of **Valid Sample Rates for ADCS with and without two-stage Sampling**.

	RPE	MPE	RRE	ERE
Orientation Alignment				
ID	0.0060	0.0029	1.6e-5	-
OOD	0.0535	0.0159	9.8e-5	-
Rectangle Constrained Sampling				
ID	0.0002	0.2267	2.3e-7	1.8e-7
OOD	0.0002	0.2775	2.3e-7	1.9e-7

Table 7: **Sampling results of both ID and OOD datasets** to demonstrate the generalization capability. The error metrics are reported using the third quartile (Q3) as in Table 2.

A.9 Generalization and Robustness Capability

To evaluate the generalization capability of our model, we tested the constraint values of the In-Distribution (ID) and Out-of-Distribution (OOD) data under different tasks. In Task *Orientation Alignment*, we set the relative pose to values out of distribution and the different positions of the midpoint together. In Task *Rectangle Constrained Sampling*, we tested the different bounding box sizes. As shown in Table 7, even when we input data from outside the training set, our model can still produce valid sampling results.