CHD: Coupled Hierarchical Diffusion for Long-Horizon Tasks

Ce Hao^{*1}**, Anxing Xiao**^{1}**, Zhiwei Xue**^{1}**, Harold Soh**^{*1,2}
^{1} School of Computing, National University of Singapore; ^{2} Smart Systems Institute, NUS
*Emails: cehao@u.nus.edu and harold@nus.edu.sg

Abstract: Diffusion-based planners have shown strong performance in short-horizon tasks but often fail in complex, long-horizon settings. We trace the failure to *loose coupling* between high-level (HL) sub-goal selection and low-level (LL) trajectory generation, which leads to incoherent plans and degraded performance. We propose **Coupled Hierarchical Diffusion** (CHD), a framework that models HL sub-goals and LL trajectories jointly within a unified diffusion process. A shared classifier passes LL feedback upstream so that sub-goals self-correct while sampling proceeds. This tight HL–LL coupling improves trajectory coherence and enables scalable long-horizon diffusion planning. Experiments across maze navigation, tabletop manipulation, and household environments show that CHD consistently outperforms both flat and hierarchical diffusion baselines. website

Keywords: Diffusion Planner, Long-horizon Planning, Hierarchical Planning

1 Introduction

Diffusion models have achieved strong results in image generation [1], video synthesis [2], and protein modeling [3]. Recently, they've been applied to robot control, where diffusion-based planners generate smooth, coherent, and multi-modal trajectories [4, 5, 6, 7]. However, as planning horizons grow, trajectory variance increases, uncertainty compounds, and achieving high reward becomes more difficult [8]. Hierarchical diffusion planners offer a promising approach towards addressing this problem by decomposing planning into high-level (HL) subgoal inference and low-level (LL) goal-conditioned trajectory generation [9, 10]—this decomposition reduces horizon length and distribution complexity.

However, a key limitation of existing baseline hierarchical diffusion (BHD) methods is the loose coupling between the HL and LL planners. HL subgoals are generated *independently* and remain *fixed*, which prevents the LL planner from refining subgoals based on trajectory outcomes [11, 9]. This disconnect limits coordinated planning and can lead to infeasible or sub-optimal trajectories. This raises a central question: *Can we couple the HL and LL planners to enable joint generation and refinement?*

In response, we propose the **C**oupled **H**ierarchical **D**iffusion (**CHD**) algorithm for long-horizon planning (Fig. 1). CHD is motivated by both practical considerations and supporting theoretical analysis (Appendix D.7). We begin with a joint diffusion model (JDM), which jointly generates multiple variables, and adapt it into CHD by introducing three key innovations: a coupled hierarchical classifier that enables bidirectional interaction between HL subgoals and LL trajectories; an asynchronous parallel generation strategy that accelerates sampling by decoupling serial dependencies; and segment-wise generation, which reduces the planning horizon and data complexity through hierarchical decomposition.

We evaluate CHD across challenging long-horizon tasks in both simulation and real-world settings. In maze navigation [12], CHD aligns HL subgoals with LL trajectory segments more effectively than both the vanilla Diffuser [4] and the baseline SHD [11], leading to more efficient paths. In a robot task planning benchmark [13], CHD successfully plans over 90 subtasks in a complex meal preparation scenario, handling high variance and sparse rewards, and outperforming strong baselines including Transformers [14], LLMs [15], and SHD.

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

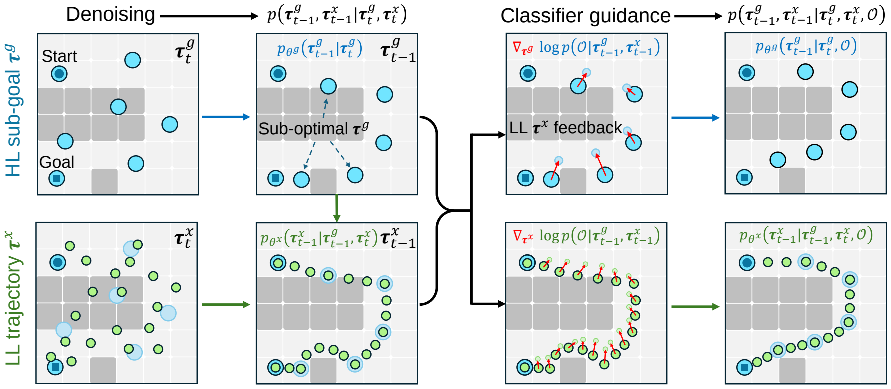

Denoising
$p(\tau^g_{t-1}, \tau^x_{t-1} | \tau^g_t, \tau^x_t)$
Classifier guidance
$p(\tau^g_{t-1}, \tau^x_{t-1} | \tau^g_t, \tau^x_t, \mathcal{O})$
HL sub-goal τ^g
LL trajectory τ^x
Start
Goal
τ^g_t
$p_{\theta^g}(\tau^g_{t-1} | \tau^g_t)$
τ^g_{t-1}
Sub-optimal τ^g
$\nabla_{\tau^g} \log p(\mathcal{O} | \tau^g_{t-1}, \tau^x_{t-1})$
LL τ^x feedback
$p_{\theta^g}(\tau^g_{t-1} | \tau^g_t, \mathcal{O})$
τ^x_t
$p_{\theta^x}(\tau^x_{t-1} | \tau^g_{t-1}, \tau^x_t)$
τ^x_{t-1}
$\nabla_{\tau^x} \log p(\mathcal{O} | \tau^g_{t-1}, \tau^x_{t-1})$
$p_{\theta^x}(\tau^x_{t-1} | \tau^g_{t-1}, \tau^x_t, \mathcal{O})$

Figure 1: Illustration of our Coupled Hierarchical Diffusion (CHD). **Left**: CHD generates the joint distribution of HL and LL through the denoising process. The HL subgoals may appear reasonable, but the resulting LL trajectories are sub-optimal. **Right**: With the coupled classifier, CHD enables LL feedback to refine sub-optimal HL subgoals, leading to improved coherence and performance.

Finally, in real-world household settings, CHD demonstrates practical viability by generating feasible subgoals and actions for both articulated and mobile manipulation tasks. To summarize, this paper makes the following contributions:

- We introduce **Coupled Hierarchical Diffusion (CHD)**, a novel algorithm that jointly generates HL subgoals and LL trajectories for long-horizon planning.
- We propose a hierarchical classifier-guided diffusion process that enables LL feedback, supports asynchronous parallel generation, and reduces complexity through segment-wise planning.
- We demonstrate CHD's effectiveness across simulated and real-world tasks, showing consistent improvements over strong baselines and highlighting the importance of HL–LL coupling.

2 Preliminaries

2.1 Problem Formulation

We focus on finite-horizon, maximum-reward trajectory optimization in a discrete-time system [4]. Let $s_k \in \mathcal{S}$ and $a_k \in \mathcal{A}$ denote the state and action at time step $k = 0, \ldots, H$, with dynamics $s_{k+1} = f(s_k, a_k)$. During planning, we aim to generate a trajectory $\boldsymbol{\tau} = (s_0, a_0, \ldots, s_H, a_H)$ that maximizes the cumulative reward $\mathcal{J}(\boldsymbol{\tau}) = \sum_{k=0}^{H} r(s_k, a_k)$.

2.2 Diffusion Models as Planners

Diffusion probabilistic models [1] are generative models that synthesize data by iteratively denoising noisy samples from T to 0. They can also be used to generate trajectories $\boldsymbol{\tau}$ [4], where the forward process $q(\boldsymbol{\tau}_t | \boldsymbol{\tau}_{t-1})$ progressively corrupts the data ("clean" trajectories), while the reverse process $p_\theta(\boldsymbol{\tau}_{t-1} | \boldsymbol{\tau}_t)$ denoises the data. The induced marginal distribution over trajectories is $p_\theta(\boldsymbol{\tau}_0) = \int p(\boldsymbol{\tau}_T) \prod_{t=1}^{T} p_\theta(\boldsymbol{\tau}_{t-1} | \boldsymbol{\tau}_t) d\boldsymbol{\tau}_{1:T}$,

where $p(\boldsymbol{\tau}_T)$ is a Gaussian prior and $\boldsymbol{\tau}_0$ represents the noiseless data. The forward process $q(\boldsymbol{\tau}_t | \boldsymbol{\tau}_{t-1})$ is typically fixed and defined by Gaussian noise addition at each time-step. Each step of the reverse process is also parameterized by a Gaussian, $p_\theta(\boldsymbol{\tau}_{t-1} | \boldsymbol{\tau}_t) = \mathcal{N}(\boldsymbol{\tau}_{t-1}; \mu_\theta(\boldsymbol{\tau}_t, t), \Sigma_\theta(\boldsymbol{\tau}_t, t))$,

where the parameters θ are optimized by minimizing a variational bound on the negative log-likelihood, $\theta^* = \arg\min_\theta -\mathbb{E}_{\boldsymbol{\tau}_0}[\log p_\theta(\boldsymbol{\tau}_0)]$.

2.3 Hierarchical Diffusion Planners

In long-horizon planning, the trajectory can be naturally decomposed into a hierarchical structure [16]. Specifically, we divide the full trajectory into N segments, each with a shorter horizon h, such that $hN = H$. The high-level (HL) planner generates a sequence of subgoals $\boldsymbol{\tau}^g = \{g_i\}_{i=1}^{N} = (g_1, g_2, \ldots, g_N)$, where g_i corresponds to the goal for the i-th segment. Let $x = (s, a)$ denote state-action pairs. The low-level (LL) trajectory segments are defined as $\boldsymbol{\tau}^x_{1:N} = \{\boldsymbol{\tau}^x_i\}_{i=1}^N$ where $\boldsymbol{\tau}^x_i = \{x_k\}_{k=(i-1)h}^{ih-1}$.

We introduce a binary optimality variable $\mathcal{O}_i = 1$ to indicate the optimality of the i-th LL segment [17]. The hierarchical planning objective is to jointly generate HL subgoals and LL trajectories when conditioned on optimality:

$$p(\boldsymbol{\tau}^g, \boldsymbol{\tau}^x_{1:N} \mid \mathcal{O}_{1:N} = 1) \propto p(\boldsymbol{\tau}^g)\, p(\boldsymbol{\tau}^x_{1:N} \mid \boldsymbol{\tau}^g)\, p(\mathcal{O}_{1:N} = 1 \mid \boldsymbol{\tau}^x_{1:N}, \boldsymbol{\tau}^g), \tag{1}$$

where $p(\boldsymbol{\tau}^g)$ defines the HL subgoal distribution, $p(\boldsymbol{\tau}^x_{1:N} \mid \boldsymbol{\tau}^g)$ defines the LL trajectory distribution conditioned on the subgoals, and $p(\mathcal{O}_{1:N} = 1 \mid \boldsymbol{\tau}^x_{1:N}, \boldsymbol{\tau}^g)$ is a classifier that guides the joint generation.

Effective hierarchical planners for long-horizon tasks should satisfy the following key properties:

P1: **Bi-directional Coupling.** High-level (HL) subgoals guide low-level (LL) trajectory generation, and LL feedback refines HL subgoals.

P2: **Parallel sampling.** HL and LL levels generate subgoals and trajectories concurrently to accelerate inference.

P3: **Reduced complexity.** Hierarchical decomposition lowers the effective planning horizon and distribution complexity, improving tractability.

However, existing hierarchical diffusion—what we call Baseline Hierarchical Diffuser (BHD) methods [9, 11]—only satisfy P3 and fail to meet P1 and P2. As shown in Fig. 2(a), during inference, BHD first generates subgoals using $p(\boldsymbol{\tau}^g)\, p(\mathcal{O}_{1:N} = 1 \mid \boldsymbol{\tau}^g)$, where $p(\boldsymbol{\tau}^g)$ is the HL diffuser and the classifier scores the subgoal sequence. It then generates LL trajectories via the conditional diffuser $p_{\theta^x}(\boldsymbol{\tau}^x_{1:N} \mid \boldsymbol{\tau}^g)$, typically implemented by subgoal inpainting. Because subgoals remain fixed before LL sampling, there is no LL to HL feedback (violating P1), and inference cannot run in parallel across levels (violating P2). Please see Appendix B for details.

3 Method: Coupled Hierarchical Diffusion

In this section, we present a planning framework that satisfies the three properties outlined above. We begin from first principles, using a canonical joint diffusion model (JDM) as our foundation (Sec. 3.1), and then adapt it to develop the Coupled Hierarchical Diffusion (CHD) planner (Sec. 3.2). To realize the desired properties, CHD incorporates three core components: coupled hierarchical classifier guidance, asynchronous parallel generation, and segment-wise generation. Due to space constraints, we focus on the key ideas and refer the reader to Appendix C and D for details.

Note that hierarchical diffusion involves two indices: the segment index $i \in [1, N]$ and the diffusion step $t \in [0, T]$. A low-level (LL) trajectory segment at step t is written as $\boldsymbol{\tau}^x_{t,i}$. For brevity, we omit the segment index and denote the full LL trajectory across segments as $\boldsymbol{\tau}^x_t = \boldsymbol{\tau}^x_{t,1:N}$.

3.1 Joint Diffusion Model (JDM)

We begin by formalizing a joint diffusion model (JDM) over high-level subgoals $\boldsymbol{\tau}^g$ and low-level trajectories $\boldsymbol{\tau}^x$, treating both as part of a single generative process. In the **forward process**, the clean pair $(\boldsymbol{\tau}^g_0, \boldsymbol{\tau}^x_0) \sim q(\boldsymbol{\tau}^g_0, \boldsymbol{\tau}^x_0)$ is corrupted independently through two separate noise processes:

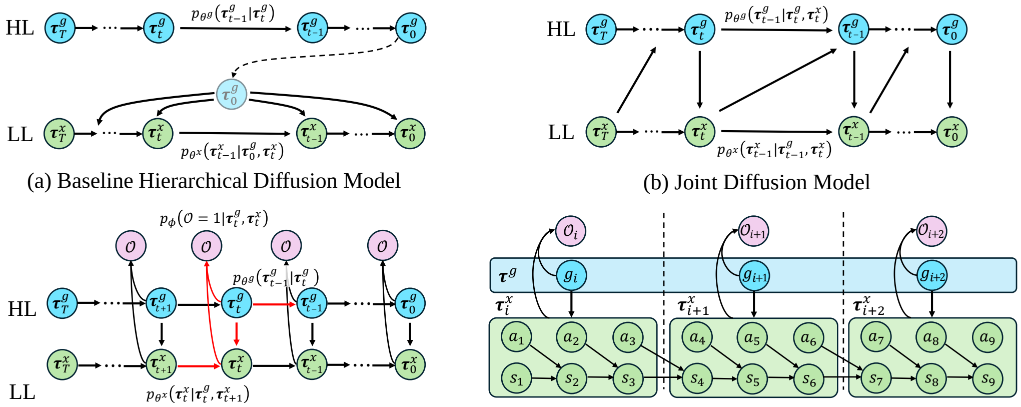
$$q(\boldsymbol{\tau}^g_{1:T}, \boldsymbol{\tau}^x_{1:T} \mid \boldsymbol{\tau}^g_0, \boldsymbol{\tau}^x_0) = q(\boldsymbol{\tau}^g_{1:T} \mid \boldsymbol{\tau}^g_0)\, q(\boldsymbol{\tau}^x_{1:T} \mid \boldsymbol{\tau}^x_0), \tag{2}$$

$$q(\boldsymbol{\tau}^\diamond_{1:T} \mid \boldsymbol{\tau}^\diamond_0) = \prod_{t=1}^{T} \mathcal{N}\left(\sqrt{1-\beta_t}\, \boldsymbol{\tau}^\diamond_{t-1},\, \beta_t \mathbf{I}^\diamond\right), \quad \diamond \in \{g, x\}. \tag{3}$$

In the **reverse process**, we model the joint generation hierarchically, as illustrated in Fig. 2(b). Applying the chain rule and Markov assumptions, the reverse factorization becomes:

$$p(\boldsymbol{\tau}^g_{0:T}, \boldsymbol{\tau}^x_{0:T}) = p(\boldsymbol{\tau}^g_T)\, p(\boldsymbol{\tau}^x_T) \prod_{t=1}^{T} p_{\theta^g}(\boldsymbol{\tau}^g_{t-1} \mid \boldsymbol{\tau}^g_t, \boldsymbol{\tau}^x_t)\, p_{\theta^x}(\boldsymbol{\tau}^x_{t-1} \mid \boldsymbol{\tau}^g_{t-1}, \boldsymbol{\tau}^x_t), \tag{4}$$

where the initial noisy variables are drawn from Gaussian priors: $p(\boldsymbol{\tau}^g_T) = \mathcal{N}(\mathbf{0}, \mathbf{I}^g)$, $p(\boldsymbol{\tau}^x_T) = \mathcal{N}(\mathbf{0}, \mathbf{I}^x)$. The reverse process is implemented through two coupled denoising models: p_{θ^g} for HL


HL
LL
$p_{\theta^g}(\tau^g_{t-1}|\tau^g_t)$
$p_{\theta^x}(\tau^x_{t-1}|\tau^g_0,\tau^x_t)$
$p_{\theta^g}(\tau^g_{t-1}|\tau^g_t,\tau^x_t)$
$p_{\theta^x}(\tau^x_{t-1}|\tau^g_{t-1},\tau^x_t)$
$p_\phi(\mathcal{O}=1|\tau^g_t,\tau^x_t)$
$p_{\theta^g}(\tau^g_{t-1}|\tau^g_t)$
$p_{\theta^x}(\tau^x_t|\tau^g_t,\tau^x_{t+1})$
τ^g
τ^x_i
τ^x_{i+1}
τ^x_{i+2}
$\mathcal{O}_i$
$\mathcal{O}_{i+1}$
$\mathcal{O}_{i+2}$
g_i
g_{i+1}
g_{i+2}
a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9
s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9

(a) Baseline Hierarchical Diffusion Model

(b) Joint Diffusion Model

(c) Coupled Hierarchical Diffusion Planner

(d) Segment-wise generation

Figure 2: CHD overcomes key limitations in hierarchical diffusion planning. (a) **BHD** plans HL subgoals and LL trajectories separately, lacking feedback and parallelism. (b) **JDM** enables tight HL and LL coupling but requires full joint-space diffusion. (c) **CHD** introduces classifier-guided LL to HL feedback and supports asynchronous, parallel generation. (d) Segment-wise generation further reduces horizon and complexity via localized planning.

subgoals and p_{θ^x} for LL trajectories. Both are conditioned on one another, forming an entangled, fully joint model.

JDM is a principled and expressive formulation that naturally provides bidirectional coupling (P1) between high-level and low-level components. However, because it diffuses over the entire joint space, it does not reduce the planning horizon or data complexity (violating P3), nor does it allow independent, parallel sampling of HL and LL (violating P2), which limits its practical scalability. Therefore, evaluating JDM is equivalent to the flat diffusion planner.

3.2 Coupled Hierarchical Diffusion Planner

We propose the Coupled Hierarchical Diffusion (CHD) planner, designed to satisfy all three properties (P1–P3). CHD builds on the Joint Diffusion Model (JDM) but introduces a key simplification: we remove the direct dependency of the HL reverse model on the LL trajectory. Specifically, we model the reverse process shown in Fig. 2(c), where the high-level reverse step is simplified to $p_{\theta^g}(\boldsymbol{\tau}^g_{t-1} \mid \boldsymbol{\tau}^g_t)$. The full reverse process is defined as:

$$p(\boldsymbol{\tau}^g_{0:T}, \boldsymbol{\tau}^x_{0:T}) = p(\boldsymbol{\tau}^g_T)\, p(\boldsymbol{\tau}^x_T) \prod_{t=1}^{T} p_{\theta^g}(\boldsymbol{\tau}^g_{t-1} \mid \boldsymbol{\tau}^g_t)\, p_{\theta^x}(\boldsymbol{\tau}^x_{t-1} \mid \boldsymbol{\tau}^g_{t-1}, \boldsymbol{\tau}^x_t), \tag{5}$$

$$p_{\theta^g}(\boldsymbol{\tau}^g_{t-1} \mid \boldsymbol{\tau}^g_t) = \mathcal{N}\left(\boldsymbol{\tau}^g_{t-1};\, \mu_{\theta^g}(\boldsymbol{\tau}^g_t, t),\, \Sigma_{\theta^g}(\boldsymbol{\tau}^g_t, t)\right), \tag{6}$$

$$p_{\theta^x}(\boldsymbol{\tau}^x_{t-1} \mid \boldsymbol{\tau}^g_{t-1}, \boldsymbol{\tau}^x_t) = \mathcal{N}\left(\boldsymbol{\tau}^x_{t-1};\, \mu_{\theta^x}(\boldsymbol{\tau}^x_t, \boldsymbol{\tau}^g_{t-1}, t),\, \Sigma_{\theta^x}(\boldsymbol{\tau}^x_t, \boldsymbol{\tau}^g_{t-1}, t)\right), \tag{7}$$

where θ^g and θ^x are the parameters of the HL and LL denoising processes, respectively.

Removing the direct dependence of the HL kernel on the LL state yields a more modular structure that we augment below with classifier guidance, asynchronous sampling, and segment-wise planning to satisfy P1–P3.

Coupled hierarchical classifier guidance achieves bidirectional coupling (P1). Embedding CHD in the hierarchical objective of Eq. (1) and applying classifier guidance yields,

$$\begin{aligned} p(\boldsymbol{\tau}^g_{0:T}, \boldsymbol{\tau}^x_{0:T} \mid \mathcal{O}_{1:N} = 1) &\propto p(\boldsymbol{\tau}^g_{0:T}, \boldsymbol{\tau}^x_{0:T})\, p_\phi(\mathcal{O}_{1:N} = 1 \mid \boldsymbol{\tau}^g_{0:T}, \boldsymbol{\tau}^x_{0:T}) \\ &= p(\boldsymbol{\tau}^g_T)\, p(\boldsymbol{\tau}^x_T) \prod_{t=1}^{T} p_{\theta^g}(\boldsymbol{\tau}^g_{t-1} \mid \boldsymbol{\tau}^g_t)\, p_{\theta^x}(\boldsymbol{\tau}^x_{t-1} \mid \boldsymbol{\tau}^g_{t-1}, \boldsymbol{\tau}^x_t)\, p_\phi(\mathcal{O}_{1:N} = 1 \mid \boldsymbol{\tau}^g_{t-1}, \boldsymbol{\tau}^x_{t-1}), \end{aligned} \tag{8}$$

where $p_\phi(\mathcal{O}_{1:N} = 1 \mid \boldsymbol{\tau}^g_t, \boldsymbol{\tau}^x_t)$ is the coupled hierarchical classifier with parameters ϕ.

As illustrated in Fig. 2(c), this classifier creates a feedback channel: HL subgoals guide LL generation, while LL trajectories offer feedback through the classifier term. The latter adjusts the HL subgoals at every reverse step to improve optimality. This mutual influence realises the desired bidirectional coupling (P2) and better approximates the full-joint model (JDM) with higher optimality compared to BHD (see theoretical arguments in Appendix D.7).

Asynchronous parallel generation enables parallel sampling (P2). In hierarchical planners, sampling both HL and LL levels in parallel substantially accelerates inference. However, in CHD, the probabilistic graph structure does not naturally support synchronous parallel generation because the LL reverse process depends on the HL trajectory, i.e., $\boldsymbol{\tau}_t^x \sim p_{\theta^x}(\boldsymbol{\tau}_t^x \mid \boldsymbol{\tau}_t^g, \boldsymbol{\tau}_{t+1}^x)$.

To enable parallelism, CHD adopts an asynchronous generation schedule by reorganizing the reverse process in Eq. (8) as:

$$\begin{aligned} &p\left(\boldsymbol{\tau}_{0:T}^g, \boldsymbol{\tau}_{0:T}^x \mid \mathcal{O}_{1:N}=1\right) \propto p(\boldsymbol{\tau}_T^g) p(\boldsymbol{\tau}_T^x) \underbrace{p_{\theta^g}(\boldsymbol{\tau}_{T-1}^g | \boldsymbol{\tau}_T^g)}_{\mathcal{P}_T^g} \\ &\cdot \prod_{t=1}^{T-1} \left[\underbrace{p_{\theta^g}(\boldsymbol{\tau}_{t-1}^g | \boldsymbol{\tau}_t^g) p_{\theta^x}(\boldsymbol{\tau}_t^x | \boldsymbol{\tau}_t^g, \boldsymbol{\tau}_{t+1}^x) p_\phi(\mathcal{O}_{1:N}=1 | \boldsymbol{\tau}_t^g, \boldsymbol{\tau}_t^x)}_{\mathcal{P}_{t-1}^{g,x}} \right] \underbrace{p_{\theta^x}(\boldsymbol{\tau}_0^x | \boldsymbol{\tau}_0^g, \boldsymbol{\tau}_1^x) p_\phi(\mathcal{O}_{1:N}=1 | \boldsymbol{\tau}_0^g, \boldsymbol{\tau}_0^x)}_{\mathcal{P}_0^x} \end{aligned} \tag{9}$$

This decomposition separates the reverse process into three stages:

1. **Initialization:** ($\mathcal{P}_T^g$): Sample $\boldsymbol{\tau}_T^g$ and $\boldsymbol{\tau}_T^x$ from Gaussian priors, then compute the HL update $\boldsymbol{\tau}_{T-1}^g$, creating an initial stagger.
2. **Asynchronous core** ($\mathcal{P}_{t-1}^{g,x}$): For each step $t = 1, \ldots, T-1$, jointly sample $(\boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x)$ by updating them in parallel. The LL trajectory depends on $\boldsymbol{\tau}_{t+1}^x$ and $\boldsymbol{\tau}_t^g$, while the HL subgoal is updated independently given $\boldsymbol{\tau}_t^g$.
3. **Final LL step** ($\mathcal{P}_0^x$): Sample the last LL step $\boldsymbol{\tau}_0^x$ conditioned on $\boldsymbol{\tau}_0^g$ and $\boldsymbol{\tau}_1^x$.

Fig. 2(c) illustrates this asynchronous structure, where red arrows indicate the reverse dependencies at each step. One complication that arises is that we cannot no longer directly apply classifier guidance; the joint pair $(\boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x)$ is updated at each step, but the classifier only scores $(\boldsymbol{\tau}_t^g, \boldsymbol{\tau}_t^x)$. Instead, we backpropagate the classifier score through the HL denoiser using the chain rule:

$$p_{\theta^g}(\boldsymbol{\tau}_{t-1}^g \mid \boldsymbol{\tau}_t^g)\, p_\phi(\mathcal{O} \mid \boldsymbol{\tau}_t^g, \boldsymbol{\tau}_t^x) \approx \mathcal{N}\left(\boldsymbol{\tau}_{t-1}^g;\, \mu_{\theta^g}(\boldsymbol{\tau}_t^g, t) + \lambda^g \Sigma_{\theta^g}(\boldsymbol{\tau}_t^g, t)\, \mathcal{J}^{\text{Asy}}(\boldsymbol{\tau}_t^g, \mu_{\theta^x}), \Sigma_{\theta^g}\right) \tag{10}$$

$$\mathcal{J}^{\text{Asy}}(\boldsymbol{\tau}_t^g, \mu_{\theta^x}) = \nabla_{\boldsymbol{\tau}^g} \log p_\phi(\mathcal{O} \mid \boldsymbol{\tau}^g, \boldsymbol{\tau}^x) \Big|_{\substack{\boldsymbol{\tau}^g = \boldsymbol{\tau}_t^g \\ \boldsymbol{\tau}^x = \mu_{\theta^x}}} \cdot \frac{\partial \boldsymbol{\tau}_t^g}{\partial \mu_{\theta^g}}, \tag{11}$$

where we have written $\mathcal{O}_{1:N} = 1$ as $\mathcal{O}$, λ^g is a guidance scaling factor, and $\partial \boldsymbol{\tau}_t^g / \partial \mu_{\theta^g} = \sqrt{1-\beta_t}$ assumes fixed noise in the forward process. Through this formulation, CHD enables both HL and LL to be denoised in parallel throughout most of the reverse process, which accelerates sampling while still benefiting from classifier guidance.

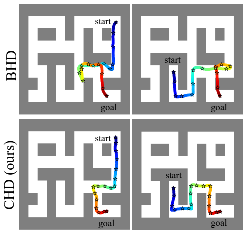
Segment-wise generation reduces planning complexity (P3). In long-horizon planning, data complexity often arises from long, multi-task trajectories. CHD mitigates this by decomposing the low-level (LL) planning process into shorter, independent segments. This approach of segment-wise generation [9] reduces the effective planning horizon and simplifies both the LL denoiser and the classifier. Specifically, we partition the LL trajectory into N segments, each of horizon h, and assume conditional independence across segments. The LL reverse model and classifier factorize as:

$$p_{\theta^x}(\boldsymbol{\tau}_{t-1,1:N}^x \mid \boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_{t,1:N}^x) = \prod_{i=1}^N p_{\theta^x}(\boldsymbol{\tau}_{t-1,i}^x \mid g_{t-1,i}, \boldsymbol{\tau}_{t,i}^x) \tag{12}$$

$$p(\mathcal{O}_{1:N} = 1 \mid \boldsymbol{\tau}_t^g, \boldsymbol{\tau}_{t,1:N}^x) = \prod_{i=1}^N p_\phi(\mathcal{O}_i = 1 \mid g_{t,i}, \boldsymbol{\tau}_{t,i}^x) \tag{13}$$

Segment independence is an approximation, but one that significantly reduces data complexity and improves tractability in long-horizon settings.

We have focused on CHD's core innovations above. Appendix D details the architecture and training setup. We also explain why JDM is unsuitable for hierarchical planning (Appendix D.6) and provide a theoretical comparison of CHD and BHD (Appendix D.7).



Environment		Diffuser	DD	BHD	HMDI	SHD	**CHD** (ours)
Maze2D	U-Maze	113.9 ± 3.1	116.2 ± 2.7	118.5 ± 5.4	120.1 ± 2.5	128.4 ± 3.6	**142.8** ± 2.9
Maze2D	Medium	121.5 ± 2.7	122.3 ± 2.1	127.1 ± 5.3	121.8 ± 1.6	135.6 ± 3.0	**149.3** ± 3.3
Maze2D	Large	123.0 ± 6.4	125.9 ± 1.6	129.0 ± 8.5	128.6 ± 2.9	155.8 ± 2.5	**179.1** ± 4.7
Single-task Average		119.5	121.5	124.9	123.5	139.9	**157.1**
Multi2D	U-Maze	128.9 ± 1.8	128.2 ± 2.1	145.0 ± 2.8	131.3 ± 1.8	144.1 ± 1.2	**149.5** ± 2.3
Multi2D	Medium	127.2 ± 3.4	127.2 ± 3.4	130.3 ± 4.7	131.6 ± 1.9	140.2 ± 1.6	**159.9** ± 4.1
Multi2D	Large	132.1 ± 5.8	130.5 ± 4.2	150.8 ± 6.0	135.4 ± 2.2	165.5 ± 0.6	**187.4** ± 4.8
Multi-task Average		129.4	129.5	142.0	132.8	149.9	**165.6**

Figure 3: **Long-horizon trajectory planning in maze navigation**. **Left**: Comparision of planned trajectories, ★ represents sub-goals. **Right**: Normalized rewards in Maze2D environments in D4RL. CHD results are calculated over 150 seeds.

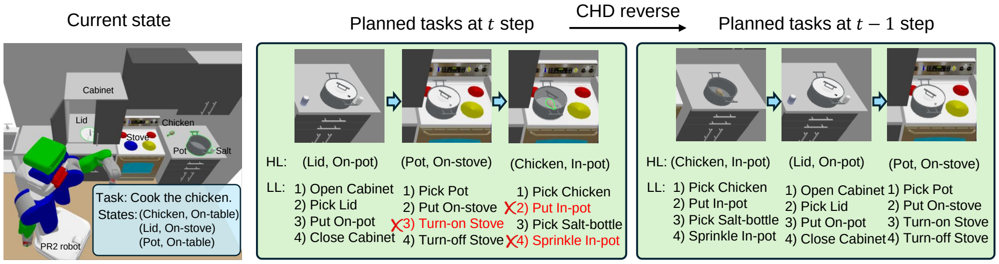


Figure 4: **Task planning experiments in Kitchen World**. Given the current state, CHD plans tasks with HL subgoal states and LL actions. During the joint reverse process, CHD can adjust the HL subgoals according to the LL actions.

4 Experiments

Our experiments evaluate CHD on long-horizon planning tasks, addressing three key questions: **(1)** Does CHD improve trajectory optimization performance? **(2)** Does CHD enhance hierarchical planning through LL feedback and reduced planning horizons? **(3)** How computationally efficient is CHD with parallel generation? We primarily test CHD on maze navigation (Sec. 4.1) and robot task planning (Sec. 4.2), and describe a real-robot case-study involving manipulation tasks (Sec. 4.4). Please see Appendix E for details on the experimental setup and implementation.

4.1 Maze Navigation

We begin with the Maze Navigation benchmark [12], where trajectories lie in continuous 2D space and HL subgoals correspond to key intermediate joint states between LL segments. The agent receives a reward of $+1$ upon reaching the target and 0 otherwise; thus, the optimal policy reaches the goal in the fewest steps. This task is challenging due to sparse rewards and sub-optimal demonstration data. We compare CHD against baseline diffusion planners, including Diffuser [4], Decision Diffuser (DD) [5] with classifier-free guidance, and baseline hierarchical diffusion methods (BHD), HMDI (BHD with graph-search improved subgoals) [11] and SHD (BHD with evenly divided subgoals) [9].

Results. CHD outperforms both flat and hierarchical diffusion baselines by 10–15% in reward across different maze settings (Fig. 3, right). This improvement reflects stronger imitation and trajectory optimization performance. Fig. 3 (left) offers a qualitative explanation for the performance gain: in BHD, subgoals are planned independently and often placed at sub-optimal corners, leading to redundant steps and collisions. In contrast, CHD couples HL and LL planning, which adapts subgoals to trajectory segments and reduces overall path length. (More details are in Appendix E.1.)

4.2 Robot Task Planning

We evaluated CHD in a long-horizon robot task planning benchmark using the Kitchen-World environment [15], where a mobile dual-arm robot must plan a sequence of actions to "cook meals" under kinematic constraints. The scene contains 20 rigid and articulated objects with randomized initial placements. State and action spaces are discretized into semantic tokens (e.g., `(Pot, In-Cabinet)`, `(Pick, Bowl)`). We collected sub-optimal demonstrations for both single-target and multi-target

Table 1: Robot Task Planning Results

Environment		VLM	Trans.	Diffuser	BHD	CHD
Single	Easy	4.71± 0.8	5.03± 0.8	6.34± 1.4	6.94± 1.3	**8.01**± 1.6
Single	Med.	3.87± 0.9	4.37± 0.7	3.51± 1.6	5.35± 1.9	**5.73**± 1.4
Single	Hard	3.54± 1.1	4.03± 0.9	2.43± 1.9	4.17± 1.7	**5.10**± 1.6
Average		3.87	4.48	4.09	5.49	**6.28**
Multi	Easy	4.45± 0.8	4.92± 0.8	5.85± 1.3	6.81± 1.5	**7.34**± 1.4
Multi	Med.	3.89± 0.8	4.45± 0.7	3.14± 1.5	4.93± 1.2	**5.65**± 1.7
Multi	Hard	3.41± 1.0	4.02± 1.0	2.08± 1.8	3.84± 1.4	**4.52**± 1.6
Average		3.91	4.46	3.69	5.19	**5.84**

[*] **Number of completed tasks (max** 10**)** ↑. Results are averaged over 1000 trials.

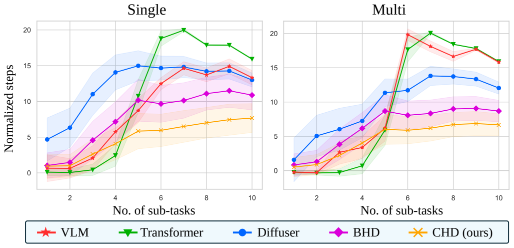


Figure 5: Normalized cumulative steps of subtasks ↓.

tasks. These were grouped by average trajectory length into Easy (50 steps), Medium (70), and Hard (90), and further segmented into 10 sub-tasks for evaluation.

We compared CHD against several learning-based planners, including a general-purpose large language model ChatGPT-4o (VLM), Transformer, Diffuser, and BHD. For diffusion models, we applied Bits Diffusion [18] for token generation. Performance was measured by the number of completed tasks and the normalized number of steps to complete sub-tasks.

Results. CHD achieved the highest task completion rate (Table 1), outperforming all baselines. Unlike VLMs and Transformers, which use auto-regressive generation and often suffer from "repetition traps" due to context ambiguity [19, 20]—diffusion-based planners iteratively refine sequences via denoising, mitigating this issue. Diffuser outperformed VLM and Transformer, and BHD's hierarchical structure provided further gains, particularly in medium and hard settings. However, BHD's fixed subgoals limited recovery when early decisions became infeasible. CHD overcame this limitation by incorporating LL feedback to refine HL subgoals during planning. Fig. 4 illustrates how CHD dynamically adjusts subgoals when infeasible actions are detected, improving overall robustness. Fig. 5 shows the normalized step count across sub-tasks. Transformer and VLM performed well on early sub-tasks but failed in later ones due to repetition traps. Among diffusion-based methods, CHD achieved the lowest overall step count and consistently outperformed others across both short- and long-horizon tasks. (More details are in Appendix E.2)

4.3 Analysis and Discussion

The above results show that **CHD consistently outperforms both flat and hierarchical baselines across long-horizon planning tasks**. It achieves higher success rates, better trajectory efficiency, and greater robustness in both continuous and discrete action spaces. In this section, we analyze the key design choices that contribute to these gains and examine the computational benefits of CHD's parallel generation.

LL feedback via coupled classifier guidance and segment-wise generation is critical for hierarchical planning. We conducted three ablation studies (Table 2) to assess CHD's components. In the first two, we removed classifier guidance or used only HL-conditioned classifiers (as in BHD). Both variants saw significant drops in performance, indicating that LL feedback is important for refining HL subgoals. In the third ablation, we replaced the segment-wise classifier (Eqn. (13)) with one conditioned on full trajectories. While this modification still outperforms BHD, it reduces performance overall due to increased distribution complexity and variance associated with conditioning on long-horizon trajectories. These results highlight the importance of both coupled guidance and segment-wise structure for effective hierarchical planning.

CHD improves sampling efficiency through parallel generation. We benchmarked training and inference times on 8×NVIDIA RTX A5000 GPUs (Table 3). CHD and BHD both train significantly faster than flat Diffuser models; the hierarhical approach reduces distribution complexity, which allowed us to reduce the number of diffusion steps from 256 to 32 while maintaining performance. However, at inference time, BHD requires *sequential* generation of HL and LL trajectories, which increases sampling time. In contrast, CHD supports asynchronous parallel generation on GPUs by coupling HL and LL updates within the diffusion process. This leads to a 30–40% reduction in sampling time compared to BHD, while maintaining superior planning performance.

Table 2: Comparison Against CHD Ablations

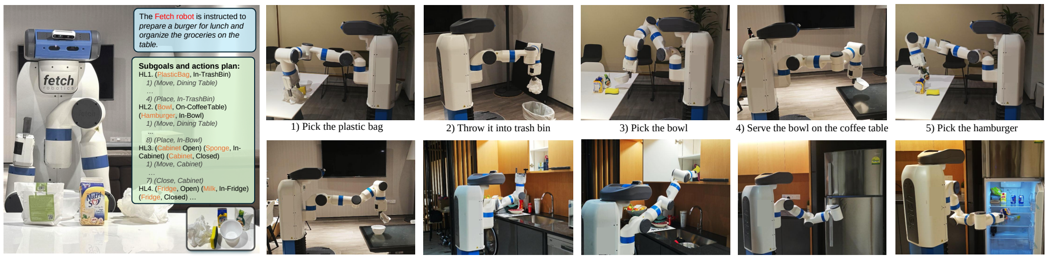
Environment	**w/o Classifier**[1]	**HL Classifier**[2]	**w/o Seg.-wise**[3]	**CHD** (ours)
Maze2D	122.5 ± 6.3	138.1 ± 3.4	142.1 ± 4.5	**157.1** ± 3.6
Multi2D	142.4 ± 4.8	146.9 ± 5.0	152.2 ± 5.3	**165.6** ± 3.7
Single-Task	4.52 ± 2.2	5.53 ± 1.7	5.77 ± 1.4	**6.28** ± 1.5
Multi-Task	4.16 ± 2.4	5.18 ± 1.5	5.42 ± 1.5	**5.84** ± 1.6

[1] Without any classifier for CHD. [2] Classifier only conditions on HL sub-goals, similar to BHD.
[3] Classifier conditions on all sub-goals and whole trajectory, not segment-wise LL classifier.
[*] Maze2D and Multi2D are average rewards in U-Maze, Medium and Large, each has 150 trials (Table 3). Single-task and Multi-task are average completed tasks of Easy, Medium and Hard, each has 10,000 trials (Table 1).

Table 3: Computation Time Comparison

Methods	Training (h)[1]		Sampling (s)[2]	
	Maze Nav.	Task Plan	Maze Nav.	Task Plan
Diffuser	15.0	5.2	1.51	1.96
BHD	5.0	1.67	0.95	1.66
CHD	5.2	1.8	0.54	1.13

We calculate wall-clock time for training till convergence and for sampling 1 planned trajectory. [1] In the training stage, diffusion models and classifiers are trained simultaneously. [2] In the sampling stage, BHD sequentially generates HL and LL, while CHD enables parallel generation.


The Fetch robot is instructed to *prepare a burger for lunch and organize the groceries on the table.*
Subgoals and actions plan:
HL1. (PlasticBag, In-TrashBin)
1) (Move, Dining Table)
...
4) (Place, In-TrashBin)
HL2. (Bowl, On-CoffeeTable) (Hamburger, In-Bowl)
1) (Move, Dining Table)
...
8) (Place, In-Bowl)
HL3. (Cabinet Open) (Sponge, In-Cabinet) (Cabinet, Closed)
1) (Move, Cabinet)
...
7) (Close, Cabinet)
HL4. (Fridge, Open) (Milk, In-Fridge) (Fridge, Closed) ...
fetch

1) Pick the plastic bag
2) Throw it into trash bin
3) Pick the bowl
4) Serve the bowl on the coffee table
5) Pick the hamburger
Fetch robot in household scenario
6) Serve the burger in the bowl
7) Open the cabinet
8) Transport the sponge to cabinet
9) Open the fridge
10) Transport the milk in fridge

Figure 6: **Real-world task-planning demonstration. Left**: The Fetch mobile robot is tasked with "*prepare a burger for lunch and organize the groceries on the table*." CHD plans over 25 HL subgoals and LL actions. **Right**: Snapshots of the robot executing planned actions in a real environment. Implementation details in Appendix E.3. (see supplementary video)

4.4 Real-Robot Demonstration

To demonstrate CHD's applicability to real-world long-horizon planning, we deployed it in a domestic service robot scenario (Fig. 6). The robot was tasked with organizing a cluttered dining table and preparing a meal. We first collected a dataset of a single-arm robot interacting with rigid and articulated household objects, similar to Sec. 4.2, and trained CHD for task planning in this domain. Follow [21], we used a mobile manipulator with model-based controllers for `pick`, `place`, and `move` actions. More complex actions like `open` and `close` were trained via imitation learning [22] using 50 demonstration trials.

Table 4: Real-Robot Planning Success Rate ↑

Tasks	**VLM**	**Trans.**	**Diffuser**	**BHD**	**CHD**
Drop bag in bin	0.85	**0.90**	**0.90**	**0.90**	**0.90**
Serve burger	0.35	0.35	0.40	0.60	**0.75**
Sponge in cabinet	0.15	0.30	0.55	0.65	**0.80**
Milk in fridge	0.20	0.25	0.30	0.50	**0.70**

To complete the entire task, the Fetch robot should complete four sub-tasks (Fig. 6). Table 4 presents the planning success rates of different methods. Given that the Fetch robot is equipped with only one arm, it must adhere to hand occupancy constraints to ensure logically coherent action sequences (e.g., opening a cabinet before picking up a sponge). However, methods such as VLM, Transformer, and Diffuser often overlook these constraints during long-horizon planning, leading to failures, particularly when the `move` action is invoked multiple times. BHD addresses this limitation by predicting subgoal states, explicitly enforcing awareness of hand occupancy constraints. CHD jointly plans HL subgoals and LL actions through coupled diffusion, resulting in significant gains in both planning robustness and downstream policy execution. This case study illustrates CHD's practical applicability and its effectiveness in learning structured, constraint-aware task plans for real-world household environments.

5 Conclusion

In this paper, we introduced Coupled Hierarchical Diffusion (CHD), a novel framework for long-horizon planning that jointly generates high-level subgoals and low-level trajectories via a coupled diffusion process. CHD addresses key limitations of existing hierarchical diffusion planners by enabling iterative feedback between planning levels, parallel sampling, and reduced complexity through segment-wise generation. Experiments across maze navigation and task planning demonstrate that CHD improves trajectory coherence, reward maximization, and sampling efficiency, establishing it as a strong candidate for scalable, long-horizon planning.

Limitations. While CHD performs well across long-horizon tasks, several limitations remain. First, it relies on manually defined sub-task segmentation; learning hierarchical structure automatically is an open challenge. Second, CHD assumes fixed subgoal lengths and uniform segment horizons, which may not suit tasks with varying temporal granularity. Finally, its reliance on learned policies and low-level primitives limits robustness under real-world noise and distribution shifts. Future work includes adaptive segmentation, perception integration, and safety-aware planning to improve generality and robustness.

Acknowledgment. This research/project is supported by A*STAR under its National Robotics Programme (NRP) (Award M23NBK0053) and the Ministry of Education, Singapore, under the Academic Research Fund Tier 1 (T1 251RES2515). The authors would also like to acknowledge support from Google.

References

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

[2] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet. Video diffusion models. *Advances in Neural Information Processing Systems*, 35:8633–8646, 2022.

[3] A. Campbell, J. Yim, R. Barzilay, T. Rainforth, and T. Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. *arXiv preprint arXiv:2402.04997*, 2024.

[4] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

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

[6] C. Hao, K. Lin, Z. Xue, S. Luo, and H. Soh. Disco: Language-guided manipulation with diffusion policies and constrained inpainting. *IEEE Robotics and Automation Letters*, 2025.

[7] X. Zhai and C. Hao. Vfp: Variational flow-matching policy for multi-modal robot manipulation. *arXiv preprint arXiv:2508.01622*, 2025.

[8] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4(1):265–293, 2021.

[9] C. Chen, F. Deng, K. Kawaguchi, C. Gulcehre, and S. Ahn. Simple hierarchical planning with diffusion. *arXiv preprint arXiv:2401.02644*, 2024.

[10] Z. Dong, J. Hao, Y. Yuan, F. Ni, Y. Wang, P. Li, and Y. Zheng. Diffuserlite: Towards real-time diffusion planning. *arXiv preprint arXiv:2401.15443*, 2024.

[11] W. Li, X. Wang, B. Jin, and H. Zha. Hierarchical diffusion for offline decision making. In *International Conference on Machine Learning*, pages 20035–20064. PMLR, 2023.

[12] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[13] Z. Yang, C. R. Garrett, T. Lozano-Pérez, L. Kaelbling, and D. Fox. Sequence-based plan feasibility prediction for efficient task and motion planning. *arXiv preprint arXiv:2211.01576*, 2022.

[14] J. Clinton and R. Lieck. Planning transformer: Long-horizon offline reinforcement learning with planning tokens. *arXiv preprint arXiv:2409.09513*, 2024.

[15] Z. Yang, C. Garrett, D. Fox, T. Lozano-Pérez, and L. P. Kaelbling. Guiding long-horizon task and motion planning with vision language models. *arXiv preprint arXiv:2410.02193*, 2024.

[16] S. Nasiriany, V. Pong, S. Lin, and S. Levine. Planning with goal-conditioned policies. *Advances in neural information processing systems*, 32, 2019.

[17] S. Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

[18] T. Chen, R. Zhang, and G. Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning. *arXiv preprint arXiv:2208.04202*, 2022.

[19] T. Hiraoka and K. Inui. Repetition neurons: How do language models produce repetitions? *arXiv preprint arXiv:2410.13497*, 2024.

[20] W. Wang, Z. Li, D. Lian, C. Ma, L. Song, and Y. Wei. Mitigating the language mismatch and repetition issues in llm-based machine translation via model editing. *arXiv preprint arXiv:2410.07054*, 2024.

[21] A. Xiao, N. Janaka, T. Hu, A. Gupta, K. Li, C. Yu, and D. Hsu. Robi butler: Remote multimodal interactions with household robot assistant. *arXiv preprint arXiv:2409.20548*, 2024.

[22] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[23] Z. Zhao, S. Cheng, Y. Ding, Z. Zhou, S. Zhang, D. Xu, and Y. Zhao. A survey of optimization-based task and motion planning: From classical to learning approaches. *IEEE/ASME Transactions on Mechatronics*, 2024.

[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] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[26] Y. Ze, G. Zhang, K. Zhang, C. Hu, M. Wang, and H. Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *ICRA 2024 Workshop on 3D Visual Representations for Robot Manipulation*, 2024.

[27] T.-W. Ke, N. Gkanatsios, and K. Fragkiadaki. 3d diffuser actor: Policy diffusion with 3d scene representations. *arXiv preprint arXiv:2402.10885*, 2024.

[28] G. Yan, Y.-H. Wu, and X. Wang. Dnact: Diffusion guided multi-task 3d policy learning. *arXiv preprint arXiv:2403.04115*, 2024.

[29] C. Hao, K. Lin, S. Luo, and H. Soh. Language-guided manipulation with diffusion policies and constrained inpainting. *arXiv preprint arXiv:2406.09767*, 2024.

[30] H. Ha, P. Florence, and S. Song. Scaling up and distilling down: Language-guided robot skill acquisition. In *Conference on Robot Learning*, pages 3766–3777. PMLR, 2023.

[31] 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.

[32] H. Zhao, X. Han, Z. Zhu, M. Liu, Y. Yu, and W. Zhang. Diffusion-based dynamics models for long-horizon rollout in offline reinforcement learning. *arXiv preprint arXiv:2405.19189*, 2024.

[33] Y. Luo, C. Sun, J. B. Tenenbaum, and Y. Du. Potential based diffusion motion planning. *arXiv preprint arXiv:2407.06169*, 2024.

[34] W. Xiao, T.-H. Wang, C. Gan, and D. Rus. Safediffuser: Safe planning with diffusion probabilistic models. *arXiv preprint arXiv:2306.00148*, 2023.

[35] S. Huang, Z. Wang, P. Li, B. Jia, T. Liu, Y. Zhu, W. Liang, and S.-C. Zhu. Diffusion-based generation, optimization, and planning in 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16750–16761, 2023.

[36] H. He, C. Bai, K. Xu, Z. Yang, W. Zhang, D. Wang, B. Zhao, and X. Li. Diffusion model is an effective planner and data synthesizer for multi-task reinforcement learning. *Advances in neural information processing systems*, 36:64896–64917, 2023.

[37] C.-F. Yang, H. Xu, T.-L. Wu, X. Gao, K.-W. Chang, and F. Gao. Planning as in-painting: A diffusion-based embodied task planning framework for environments under uncertainty. *arXiv preprint arXiv:2312.01097*, 2023.

[38] H. Nisonoff, J. Xiong, S. Allenspach, and J. Listgarten. Unlocking guidance for discrete state-space diffusion and flow models. *arXiv preprint arXiv:2406.01572*, 2024.

[39] Z. Zhu, H. Zhao, H. He, Y. Zhong, S. Zhang, H. Guo, T. Chen, and W. Zhang. Diffusion models for reinforcement learning: A survey. *arXiv preprint arXiv:2311.01223*, 2023.

[40] T. Zhang, J. Guan, L. Zhao, Y. Li, D. Li, Z. Zeng, L. Sun, Y. Chen, X. Wei, L. Li, et al. Preferred-action-optimized diffusion policies for offline reinforcement learning. *arXiv preprint arXiv:2405.18729*, 2024.

[41] Z. Wang, J. J. Hunt, and M. Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022.

[42] A. Z. Ren, J. Lidard, L. L. Ankile, A. Simeonov, P. Agrawal, A. Majumdar, B. Burchfiel, H. Dai, and M. Simchowitz. Diffusion policy policy optimization. *arXiv preprint arXiv:2409.00588*, 2024.

[43] Y. Wang, L. Wang, Y. Jiang, W. Zou, T. Liu, X. Song, W. Wang, L. Xiao, J. Wu, J. Duan, et al. Diffusion actor-critic with entropy regulator. *arXiv preprint arXiv:2405.15177*, 2024.

[44] M. Psenka, A. Escontrela, P. Abbeel, and Y. Ma. Learning a diffusion model policy from rewards via q-score matching. *arXiv preprint arXiv:2312.11752*, 2023.

[45] S. E. Ada, E. Oztop, and E. Ugur. Diffusion policies for out-of-distribution generalization in offline reinforcement learning. *IEEE Robotics and Automation Letters*, 2024.

[46] P. Hansen-Estruch, I. Kostrikov, M. Janner, J. G. Kuba, and S. Levine. Idql: Implicit q-learning as an actor-critic method with diffusion policies. *arXiv preprint arXiv:2304.10573*, 2023.

[47] J. Zhang, Y. Cheng, S. Cao, and X. Wang. Offline reinforcement learning with reverse diffusion guide policy. *IEEE Transactions on Industrial Informatics*, 2024.

[48] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

[49] S. Chen, A. Xiao, and D. Hsu. Llm-state: Expandable state representation for long-horizon task planning in the open world. *arXiv preprint arXiv:2311.17406*, 2023.

[50] B. Aceituno and A. Rodriguez. A hierarchical framework for long horizon planning of object-contact trajectories. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 189–196. IEEE, 2022.

[51] K. Pertsch, O. Rybkin, F. Ebert, S. Zhou, D. Jayaraman, C. Finn, and S. Levine. Long-horizon visual planning with goal-conditioned hierarchical predictors. *Advances in Neural Information Processing Systems*, 33:17321–17333, 2020.

[52] Z. Feng, H. Luan, K. Y. Ma, and H. Soh. Diffusion meets options: Hierarchical generative skill composition for temporally-extended tasks. *arXiv preprint arXiv:2410.02389*, 2024.

[53] W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. *arXiv preprint arXiv:2409.01652*, 2024.

[54] D. Paulius, A. Agostini, and D. Lee. Long-horizon planning and execution with functional object-oriented networks. *IEEE Robotics and Automation Letters*, 8(8):4513–4520, 2023.

[55] Z. Huang, Y. Lin, F. Yang, and D. Berenson. Subgoal diffuser: Coarse-to-fine subgoal generation to guide model predictive control for robot manipulation. *arXiv preprint arXiv:2403.13085*, 2024.

[56] Z. Liang, Y. Mu, H. Ma, M. Tomizuka, M. Ding, and P. Luo. Skilldiffuser: Interpretable hierarchical planning via skill abstractions in diffusion-based task execution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16467–16476, 2024.

[57] W. K. Kim, M. Yoo, and H. Woo. Robust policy learning via offline skill diffusion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 13177–13184, 2024.

[58] S. Kim, Y. Choi, D. E. Matsunaga, and K.-E. Kim. Stitching sub-trajectories with conditional diffusion model for goal-conditioned offline rl. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 13160–13167, 2024.

[59] Z. Wu, S. Ye, M. Natarajan, and M. C. Gombolay. Diffusion-reinforcement learning hierarchical motion planning in adversarial multi-agent games. *arXiv preprint arXiv:2403.10794*, 2024.

[60] C. Zhang, D. Jiang, K. Jiang, and B. Jiang. A hierarchical multivariate denoising diffusion model. *Information Sciences*, 648:119623, 2023.

[61] H. Wang, L. Qi, B. Fang, and Y. Sun. Hierarchical visual policy learning for long-horizon robot manipulation in densely cluttered scenes. *arXiv preprint arXiv:2312.02697*, 2023.

[62] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11461–11471, 2022.

[63] P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

[64] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the international conference on automated planning and scheduling*, volume 30, pages 440–448, 2020.

[65] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023.

[66] M. Minderer, A. Gritsenko, and N. Houlsby. Scaling open-vocabulary object detection. *Advances in Neural Information Processing Systems*, 36, 2024.

[67] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[68] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

A Related Works

A.1 Diffusion Models in Robotics

Diffusion probabilistic models [1] are powerful generative frameworks that progressively denoise Gaussian distributions into target data distributions. These models have also been leveraged for imitation learning in robot policies and planners [23, 24].

Diffusion policies utilize diffusion models to generate actions imitated from human demonstrations. Diffusion Policy [25] employed vision-based diffusion models to predict actions for robot manipulation, significantly enhancing imitation learning performance in both simulation and real-world robots. Extending this, 3D diffusion policies [26, 27] incorporated 3D point cloud observations to enrich spatial relationships and further improve manipulation performance. DNAct [28] applied 3D diffusion policies in multi-task environments, demonstrating the advantage of diffusion models in multi-modal distributions. [29] utilized VLMs to generate keyframes to guide the diffusion policy. To address scalability in imitation learning, [30] utilized visual-language models to generate diverse environments, paving the way for scalable robot foundation models.

Diffusion planners generate trajectories for mobile robots and manipulation tasks [31]. Diffuser [4] pioneered the use of diffusion models to synthesize goal-conditioned trajectories. Subsequent works enhanced diffusion planners with dynamic models [32], potential fields [33], safety constraints [34], 3D observations [35], and multi-task settings [36]. Additionally, trajectory optimization via offline reinforcement learning was integrated into diffusion planners. Diffuser [4] first employed classifier guidance for reward maximization, while [37] introduced inpainting methods for planning under uncertainty. Advances like discrete flow models [38] and classifier-free guidance [5] further aligned trajectory prediction with reward optimization.

Reinforcement learning (RL) has also benefited from diffusion models, surpassing traditional MLP-based policies [39, 40]. Diffusion-QL [41] used diffusion models as value networks, achieving superior value estimation. Diffusion-PPO [42] employed diffusion models as policy networks optimized by PPO, significantly enhancing policy optimization. Similarly, Diffusion-AC [43] revolutionized RL by integrating diffusion models into actor-critic frameworks. Further enhancements included Q-Score Matching [44] and out-of-distribution generalization [45]. Offline RL integration, as seen in works like [46] and [47], validated the ability of diffusion models to handle complex multi-modal distributions in policies and value networks.

A.2 Hierarchical Trajectory Planners

Hierarchical trajectory planning addresses the challenges of long-horizon trajectory generation [48]. By decomposing trajectories into high-level (HL) subgoals and low-level (LL) segments, this approach reduces trajectory complexity, thereby improving planning efficiency and success rates. A critical challenge is maintaining coherent interactions between HL and LL levels.

Long-horizon task planning is particularly challenging in robot navigation and manipulation due to the need for both discrete grounded subtasks and fine-grained robot actions [49, 23]. Hierarchical planners are widely adopted in such scenarios [50]. Subgoal prediction [51] sequentially generates optimal subgoals and conditionally derives actions. High-level options [52], representing abstract skills, further expand the scope of long-horizon planning, enhancing generalizability and transferability. Recent developments in large language models (LLMs) and vision-language models (VLMs) have enabled task planning via general-purpose frameworks [15]. ReKep [53] leveraged VLMs to predict affordances and connections for long-horizon tasks. Hierarchical planners are also pivotal in tool-use planning, addressing interactions with functional tools and target objects [54].

Hierarchical planning with diffusion models has emerged as a promising approach. Subgoal Diffuser [55] employed diffusion models for HL subgoal generation, guiding LL model predictive controllers. SkillDiffuser [56] used VQ-VAE for discrete HL skill generation, while LL diffusion models predicted manipulation videos. Skill-diffusion [57] applied diffusion models for HL skill prediction in long-horizon tasks. Stitching Diffusion [58] utilized offline RL to combine sub-segments from diffusion models, and Option-based Diffusion [52] planned HL abstracted options to guide LL diffusion planners.

Advanced hierarchical diffusion planners use diffusion models at both HL and LL levels. HDMI [11] introduced a two-level diffusion planner, where HL subgoals were optimized by graph models, and LL segments were generated conditionally. SHD [9] simplified this approach, using separate diffusion models for HL subgoals and LL segments. This baseline hierarchical diffuser (BHD) approach, however, suffers from a critical limitation: the independence of HL and LL levels prevents adjustments to erroneous subgoals. Motivated by this, we propose the Coupled Hierarchical Diffuser (CHD). Additionally, hierarchical diffusion planners have been applied in multi-agent games [59], multi-variable generation [60], and cluttered object environments [61]. Hierarchical structures have also been leveraged to accelerate sampling speeds [10].

B Baseline Hierarchical Diffusion Planner

In Section 2, we introduced the problem formulation of maximum-reward trajectory optimization, Diffuser structure and the baseline hierarchical diffuser (BHD). In the appendix, we present the complete formulation of BHD as a comparison to our proposed coupled hierarchical diffuser (CHD) method.

B.1 Baseline Hierarchical Diffuser Structure

The Baseline Hierarchical Diffuser (BHD) is a straightforward application of diffusion models to hierarchical planning. HMDI [11] and SHD [9] are two instances of BHD. BHD operates in two stages: a high-level (HL) planner generates sub-goals, and a low-level (LL) planner produces trajectory segments that meet these sub-goals via end-point inpainting [62].

$$\begin{aligned}
&p(\boldsymbol{\tau}^g, \boldsymbol{\tau}^x \mid \mathcal{O}=1)\\
&\quad= p(\boldsymbol{\tau}^g, \boldsymbol{\tau}^x)p(\mathcal{O}=1 \mid \boldsymbol{\tau}^g, \boldsymbol{\tau}^x_{1:N})\\
&\quad= p(\boldsymbol{\tau}^g)p(\boldsymbol{\tau}^x \mid \boldsymbol{\tau}^g)p(\mathcal{O}=1 \mid \boldsymbol{\tau}^x, \boldsymbol{\tau}^g).
\end{aligned} \tag{14}$$

In Eqn. (14), BHD utilizes the Bayesian rule to decompose the control-as-inference [17] to optimize the trajectories to achieve ‘optimality’. The HL planner is a subgoal diffusion model $p(\boldsymbol{\tau}^g)$ with classifier-guidance $p(\mathcal{O}=1 \mid \boldsymbol{\tau}^x, \boldsymbol{\tau}^g)$, and the LL is a conditional diffusion model $p(\boldsymbol{\tau}^x \mid \boldsymbol{\tau}^g)$.

High-Level (HL) Planner generates a sequence of sub-goals $\boldsymbol{\tau}^g$, and we impose that these sub-goals are “optimal”, indicated by $\mathcal{O}=1$, as $p(\boldsymbol{\tau}^g)\,p(\mathcal{O}=1 \mid \boldsymbol{\tau}^g)$. Parameterizing these distributions, the HL reverse model is $p_{\theta^g}(\boldsymbol{\tau}^g_{t-1} \mid \boldsymbol{\tau}^g_t)\,p_\theta(\mathcal{O}=1 \mid \boldsymbol{\tau}^g_t)$, where $p_{\theta^g}(\cdot)$ is the generative model for high-level sub-goals and $p_\theta(\mathcal{O}=1 \mid \boldsymbol{\tau}^g_t)$ models the probability that the sub-goal at time t is optimal. Therefore, the HL planner is guided by the classifier and the reverse process is as [4]:

$$\begin{aligned}
p_{\theta^g}\left(\boldsymbol{\tau}^g_{t-1} \mid \boldsymbol{\tau}^g_t, \mathcal{O}_{1:N}\right) &\approx \mathcal{N}\left(\boldsymbol{\tau}^g_{t-1}; \mu_{\theta^g}\left(\boldsymbol{\tau}^g_t, t\right) + \Sigma^t \mathcal{J}\left(\mu_{\theta^g}\right), \Sigma^t\right)\\
\mathcal{J}\left(\mu_{\theta^g}\right) &= \nabla_{\mu_{\theta^g}} \log p\left(\mathcal{O}_{1:N}=1 \mid \boldsymbol{\tau}^g\right)|_{\boldsymbol{\tau}^g=\mu_{\theta^g}}
\end{aligned} \tag{15}$$

Low-Level (LL) Planner generate trajectories $\boldsymbol{\tau}^x$ that connect these sub-goals $\boldsymbol{\tau}^g$ while satisfying endpoint constraints. We start with the conditional distribution $p(\boldsymbol{\tau}^x \mid \boldsymbol{\tau}^g)$. The LL planner is defined by a diffusion process that refines initially noisy trajectories into coherent paths $p_{\theta^x}(\boldsymbol{\tau}^x_{t-1} \mid \boldsymbol{\tau}^x_t, \boldsymbol{\tau}^g_0)$.

B.2 Loss function

The training of HL and LL diffusion models are separated, and the variational bound of negative log-likelihood can be formulated as [11]:

$$\mathbb{E}_q[-\log p_{\theta^g}(\boldsymbol{\tau}^g_0) - \log p_{\theta^x}(\boldsymbol{\tau}^x_0)] \leq \mathbb{E}_q\left[-\log \frac{p_{\theta^g}(\boldsymbol{\tau}^g_{0:T})}{q(\boldsymbol{\tau}^g_{1:T} \mid \boldsymbol{\tau}^g_0)} - \log \frac{p_{\theta^x}(\boldsymbol{\tau}^g_{0:T} \mid \boldsymbol{\tau}^g_0)}{q(\boldsymbol{\tau}^x_{1:T} \mid \boldsymbol{\tau}^x_0)}\right] := L^{\mathrm{BHD}} \tag{16}$$

We further expand the variational bound to derive the loss function as:

$$L^{\text{BHD}} = \mathbb{E}_q \Big[\underbrace{D_{\text{KL}}\left(q(\boldsymbol{\tau}^g_T \mid \boldsymbol{\tau}^g_0) \| p(\boldsymbol{\tau}^g_T)\right)}_{L^g_T} + \underbrace{D_{\text{KL}}\left(q(\boldsymbol{\tau}^x_T \mid \boldsymbol{\tau}^x_0) \| p(\boldsymbol{\tau}^x_T)\right)}_{L^x_T} + \sum_{t=2}^{T} \underbrace{D_{\text{KL}}\left(q(\boldsymbol{\tau}^g_{t-1} \mid \boldsymbol{\tau}^g_t, \boldsymbol{\tau}^g_0) \| p_\theta(\boldsymbol{\tau}^g_{t-1} \mid \boldsymbol{\tau}^g_t)\right)}_{L^g_{t-1}}$$
$$+ \sum_{t=2}^{T} \underbrace{D_{\text{KL}}\left(q(\boldsymbol{\tau}^x_{t-1} \mid \boldsymbol{\tau}^x_t, \boldsymbol{\tau}^x_0) \| p_\theta(\boldsymbol{\tau}^x_{t-1} \mid \boldsymbol{\tau}^x_t, \boldsymbol{\tau}^g_0)\right)}_{L^x_{t-1}} \underbrace{- \log p_\theta(\boldsymbol{\tau}^g_0 \mid \boldsymbol{\tau}^g_1)}_{L^g_0} \underbrace{- \log p_\theta(\boldsymbol{\tau}^x_0 \mid \boldsymbol{\tau}^x_1, \boldsymbol{\tau}^g_1)}_{L^x_0} \Big] \tag{17}$$

In Eqn. (17), the HL $\boldsymbol{\tau}^g$ diffusion is independent, and the LL $\boldsymbol{\tau}^x$ conditional diffusion is implemented via inpainting [62]. Therefore, two diffusion models are independent in the training stage and could be trained separately.

C Joint Diffusion Model

In section 3.1, we introduce the basic formulation of the joint diffusion model(JDM). JDM is a direct expansion of one diffusion model that jointly generates two variables as two coupled diffusion processes. We provide some details in this section.

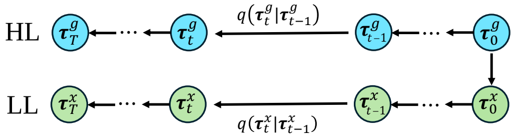

HL
$\boldsymbol{\tau}^g_T$
$\boldsymbol{\tau}^g_t$
$q(\boldsymbol{\tau}^g_t|\boldsymbol{\tau}^g_{t-1})$
$\boldsymbol{\tau}^g_{t-1}$
$\boldsymbol{\tau}^g_0$
LL
$\boldsymbol{\tau}^x_T$
$\boldsymbol{\tau}^x_t$
$q(\boldsymbol{\tau}^x_t|\boldsymbol{\tau}^x_{t-1})$
$\boldsymbol{\tau}^x_{t-1}$
$\boldsymbol{\tau}^x_0$

Figure 7: Probabilistic graph model of the forward process of joint diffusion model

Forward process. Figure 7 shows the probabilistic graph model for the forward process of JDM. We consider two cases.

When $t = 0$, the LL trajectory is conditioned on the HL subgoals as:

$$q(\boldsymbol{\tau}^g_t, \boldsymbol{\tau}^x_0) = q(\boldsymbol{\tau}^x_0|\boldsymbol{\tau}^g_0)q(\boldsymbol{\tau}^g_0);$$

when $t \geq 0$, two diffusion processes have the Markov property that

$$q(\boldsymbol{\tau}^g_t|\boldsymbol{\tau}^g_{t-1}, \boldsymbol{\tau}^x_{t-1}) = q(\boldsymbol{\tau}^g_t|\boldsymbol{\tau}^g_{t-1}), \quad q(\boldsymbol{\tau}^x_t|\boldsymbol{\tau}^g_{t-1}, \boldsymbol{\tau}^x_{t-1}) = q(\boldsymbol{\tau}^x_t|\boldsymbol{\tau}^x_{t-1}).$$

Therefore, the forward-nosing process of two joint variables is

$$\begin{aligned} q(\boldsymbol{\tau}^g_{1:T}, \boldsymbol{\tau}^x_{1:T}|\boldsymbol{\tau}^g_0, \boldsymbol{\tau}^x_0) &= \prod_{t=1}^{T} q(\boldsymbol{\tau}^g_t|\boldsymbol{\tau}^g_{t-1}, \boldsymbol{\tau}^x_{t-1})q(\boldsymbol{\tau}^x_t|\boldsymbol{\tau}^g_{t-1}, \boldsymbol{\tau}^x_{t-1}) \\ &= \prod_{t=1}^{T} q(\boldsymbol{\tau}^g_t|\boldsymbol{\tau}^g_{t-1})q(\boldsymbol{\tau}^x_t|\boldsymbol{\tau}^x_{t-1}) = q(\boldsymbol{\tau}^g_{1:T}|\boldsymbol{\tau}^g_0)q(\boldsymbol{\tau}^x_{1:T}|\boldsymbol{\tau}^x_0) \end{aligned} \tag{18}$$

$q(\boldsymbol{\tau}^g_{1:T}|\boldsymbol{\tau}^g_0)$ and $q(\boldsymbol{\tau}^x_{1:T}|\boldsymbol{\tau}^x_0)$ can be regard as two independent forward processes.

Reverse process. Figure 2(b) shows the probabilistic graph model for reverse process of JDM. We can easily expand the joint distribution using Bayes Theorem as,

$$\begin{aligned} p(\boldsymbol{\tau}^g_{0:T}, \boldsymbol{\tau}^x_{0:T}) &= p(\boldsymbol{\tau}^g_T, \boldsymbol{\tau}^x_T) \prod_{t=1}^{T} p_\theta(\boldsymbol{\tau}^g_{t-1}, \boldsymbol{\tau}^x_{t-}|\boldsymbol{\tau}^g_t, \boldsymbol{\tau}^x_t) \\ &= p(\boldsymbol{\tau}^g_T)p(\boldsymbol{\tau}^x_T) \prod_{t=1}^{T} p_{\theta^g}(\boldsymbol{\tau}^g_{t-1}|\boldsymbol{\tau}^g_t, \boldsymbol{\tau}^x_t)p_{\theta^x}(\boldsymbol{\tau}^x_{t-1}|\boldsymbol{\tau}^g_{t-1}, \boldsymbol{\tau}^g_t, \boldsymbol{\tau}^x_t) \\ &= p(\boldsymbol{\tau}^g_T)p(\boldsymbol{\tau}^x_T) \prod_{t=1}^{T} p_{\theta^g}(\boldsymbol{\tau}^g_{t-1}|\boldsymbol{\tau}^g_t, \boldsymbol{\tau}^x_t)p_{\theta^x}(\boldsymbol{\tau}^x_t|\boldsymbol{\tau}^g_{t-1}, \boldsymbol{\tau}^x_t) \end{aligned} \tag{19}$$

In $p_{\theta^x}(\boldsymbol{\tau}_t^x|\boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^g\boldsymbol{\tau}_t^x)$, the $\boldsymbol{\tau}_t^g$ is removed due to the Markov property.

Loss function. We derive the loss function with the variational bound of negative log-likelihood for the generative model as,

$$\mathbb{E}_q\left[-\log p_\theta(\boldsymbol{\tau}_0^g, \boldsymbol{\tau}_0^x)\right] \leq \mathbb{E}_q\left[-\log \frac{p_\theta(\boldsymbol{\tau}_{0:T}^g, \boldsymbol{\tau}_{0:T}^x)}{q(\boldsymbol{\tau}_{1:T}^g, \boldsymbol{\tau}_{1:T}^x|\boldsymbol{\tau}_0^g, \boldsymbol{\tau}_0^x)}\right] = L^{\text{JDM}} \tag{20}$$

Then, we bring in the forward and reverse processes to derive [1]:

$$\begin{aligned}
L^{\text{JDM}} &= \mathbb{E}_q\left[-\log \frac{p_\theta(\boldsymbol{\tau}_{0:T}^g, \boldsymbol{\tau}_{0:T}^x)}{q(\boldsymbol{\tau}_{1:T}^g, \boldsymbol{\tau}_{1:T}^x|\boldsymbol{\tau}_0^g, \boldsymbol{\tau}_0^x)}\right] \\
&= \mathbb{E}_q\left[-\log p(\boldsymbol{\tau}_T^g) - \log p(\boldsymbol{\tau}_T^x) - \sum_{t=1}^T \log \frac{p_\theta(\boldsymbol{\tau}_{t-1}^g|\boldsymbol{\tau}_t^g, \boldsymbol{\tau}_t^x)p_\theta(\boldsymbol{\tau}_{t-1}^x|\boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x)}{q(\boldsymbol{\tau}_t^g|\boldsymbol{\tau}_{t-1}^g)q(\boldsymbol{\tau}_t^x|\boldsymbol{\tau}_{t-1}^x)}\right] \\
&= \mathbb{E}_q\left[-\log p(\boldsymbol{\tau}_T^g) - \log p(\boldsymbol{\tau}_T^x) - \sum_{t=2}^T \log \frac{p_\theta(\boldsymbol{\tau}_{t-1}^g|\boldsymbol{\tau}_t^g, \boldsymbol{\tau}_t^x)p_\theta(\boldsymbol{\tau}_{t-1}^x|\boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x)}{q(\boldsymbol{\tau}_t^g|\boldsymbol{\tau}_{t-1}^g)q(\boldsymbol{\tau}_t^x|\boldsymbol{\tau}_{t-1}^x)} - \log \frac{p_\theta(\boldsymbol{\tau}_0^g|\boldsymbol{\tau}_1^g, \boldsymbol{\tau}_1^x)p_\theta(\boldsymbol{\tau}_0^x|\boldsymbol{\tau}_0^g, \boldsymbol{\tau}_1^x)}{q(\boldsymbol{\tau}_1^g|\boldsymbol{\tau}_0^g)q(\boldsymbol{\tau}_1^x|\boldsymbol{\tau}_0^x)}\right] \\
&= \mathbb{E}_q\Bigg[\underbrace{D_{\text{KL}}\big(q(\boldsymbol{\tau}_T^g|\boldsymbol{\tau}_0^g)||p(\boldsymbol{\tau}_T^g)\big)}_{L_T^g} + \underbrace{D_{\text{KL}}\big(q(\boldsymbol{\tau}_T^x|\boldsymbol{\tau}_0^x)||p(\boldsymbol{\tau}_T^x)\big)}_{L_T^x} + \sum_{t=2}^T \underbrace{D_{\text{KL}}\big(q(\boldsymbol{\tau}_t^g|\boldsymbol{\tau}_{t-1}^g)||p_\theta(\boldsymbol{\tau}_{t-1}^g|\boldsymbol{\tau}_t^g, \boldsymbol{\tau}_t^x)\big)}_{L_{t-1}^g} \\
&\quad + \sum_{t=2}^T \underbrace{D_{\text{KL}}\big(q(\boldsymbol{\tau}_t^x|\boldsymbol{\tau}_{t-1}^x, \boldsymbol{\tau}_0^x)||p_\theta(\boldsymbol{\tau}_t^x|\boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x)\big)}_{L_{t-1}^x} \underbrace{-\log p_\theta(\boldsymbol{\tau}_0^g|\boldsymbol{\tau}_1^g, \boldsymbol{\tau}_1^x)}_{L_0^g} \underbrace{-\log p_\theta(\boldsymbol{\tau}_0^x|\boldsymbol{\tau}_0^g, \boldsymbol{\tau}_1^x)}_{L_0^x}\Bigg]
\end{aligned} \tag{21}$$

In Eqn. (21), L_T^g and L_T^x are neglected since they do not have learnable parameters. L_0^g and L_0^x are the reconstruction terms that generate the original data distribution. In the experiments, we either use a Gaussian model for continuous variables or Bits Discretization [18] for discrete variables. The most important losses L_{t-1}^g and L_{t-1}^x are denoising matching terms.

D Coupled Hierarchical Diffusion Planner

In section 3.2, we propose the couple hierarchical diffusion algorithm (CHD) as an effective planner for three properties. The basic formulation of CHD is a simplification of JDM. The change is in the reverse process $p_{\theta^g}(\boldsymbol{\tau}_{t-1}^g|\boldsymbol{\tau}_t^g, \boldsymbol{\tau}_t^x)$, $\boldsymbol{\tau}_t^x$ is neglected so that the new reverse process is shown in Figure 2(c).

$$p(\boldsymbol{\tau}_{0:T}^g, \boldsymbol{\tau}_{0:T}^x) = p(\boldsymbol{\tau}_T^g)p(\boldsymbol{\tau}_T^x)\prod_{t=1}^T p_{\theta^g}(\boldsymbol{\tau}_{t-1}^g|\boldsymbol{\tau}_t^g)p_{\theta^x}(\boldsymbol{\tau}_{t-1}^x|\boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x)$$

D.1 Loss function and parametrization

We can derive the variational bound and loss function analogous to JDM (Eqn. 21) as,

$$\begin{aligned}
L^{\text{CHD}} = \mathbb{E}_q\Bigg[&\underbrace{D_{\text{KL}}\big(q(\boldsymbol{\tau}_T^g|\boldsymbol{\tau}_0^g)||p(\boldsymbol{\tau}_T^g)\big)}_{L_T^g} + \underbrace{D_{\text{KL}}\big(q(\boldsymbol{\tau}_T^x|\boldsymbol{\tau}_0^x)||p(\boldsymbol{\tau}_T^x)\big)}_{L_T^x} + \sum_{t=2}^T \underbrace{D_{\text{KL}}\big(q(\boldsymbol{\tau}_t^g|\boldsymbol{\tau}_{t-1}^g)||p_\theta(\boldsymbol{\tau}_{t-1}^g|\boldsymbol{\tau}_t^g)\big)}_{L_{t-1}^g} \\
&+ \sum_{t=2}^T \underbrace{D_{\text{KL}}\big(q(\boldsymbol{\tau}_t^x|\boldsymbol{\tau}_{t-1}^x, \boldsymbol{\tau}_0^x)||p_\theta(\boldsymbol{\tau}_t^x|\boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x)\big)}_{L_{t-1}^x} \underbrace{-\log p_\theta(\boldsymbol{\tau}_0^g|\boldsymbol{\tau}_1^g, \boldsymbol{\tau}_1^x)}_{L_0^g} \underbrace{-\log p_\theta(\boldsymbol{\tau}_0^x|\boldsymbol{\tau}_0^g, \boldsymbol{\tau}_1^x)}_{L_0^x}\Bigg]
\end{aligned} \tag{22}$$

Therefore, $p_{\theta^g}\left(\boldsymbol{\tau}_{t-1}^g \mid \boldsymbol{\tau}_t^g\right)$ and $p_{\theta^x}\left(\boldsymbol{\tau}_{t-1}^x \mid \boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x\right)$ are the HL and LL reverse diffusion models. We parameterize them as in DDPM [1].

$$p_{\theta^g}\left(\boldsymbol{\tau}_{t-1}^g \mid \boldsymbol{\tau}_t^g\right) = \mathcal{N}\left(\boldsymbol{\tau}_{t-1}^g; \mu_{\theta^g}\left(\boldsymbol{\tau}_t^g, t\right), \Sigma_{\theta^g}\left(\boldsymbol{\tau}_t^g, t\right)\right) \tag{23}$$

$$p_{\theta^x}\left(\boldsymbol{\tau}_{t-1}^x \mid \boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x\right)=\mathcal{N}\left(\boldsymbol{\tau}_{t-1}^x ; \mu_{\theta^x}\left(\boldsymbol{\tau}_t^x, \boldsymbol{\tau}_{t-1}^g, t\right), \Sigma_{\theta^x}\left(\boldsymbol{\tau}_t^x, \boldsymbol{\tau}_{t-1}^g, t\right)\right) \tag{24}$$

The exact loss function can be written as,

$$\mathcal{L}_{\text{Diff}}=\mathbb{E}_{\boldsymbol{\tau}_0^g, \boldsymbol{\tau}_0^x, \epsilon^g, \epsilon^x, t}\left[\left\|\epsilon^g-\epsilon_{\theta^g}^g(\boldsymbol{\tau}_t^g, t)\right\|+\left\|\epsilon^x-\epsilon_{\theta^x}^x(\boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x, t)\right\|\right] \tag{25}$$

Note that HL and LL can be trained with one loss function or trained separately with their own losses. While in either way, all variables $\boldsymbol{\tau}_0^g, \boldsymbol{\tau}_0^x, \epsilon^g, \epsilon^x$ must be sampled in the same demonstration. In practice, we can train HL and LL in two GPU devices for acceleration.

D.2 Coupled hierarchical classifier guidance

The coupled hierarchical classifier guidance enables LL feedback to HL because the classifier considers and adjusts both levels. We copy Eqn. (8) here and apply the classifier guidance for HL and LL diffusion models.

$$\begin{aligned}
p\left(\boldsymbol{\tau}_{0:T}^g, \boldsymbol{\tau}_{0:T}^x \mid \mathcal{O}_{1:N}=1\right) &\propto p\left(\boldsymbol{\tau}_{0:T}^g, \boldsymbol{\tau}_{0:T}^x\right) p\left(\mathcal{O}_{1:N}=1 \mid \boldsymbol{\tau}_{0:T}^g, \boldsymbol{\tau}_{0:T}^x\right) \\
&= p(\boldsymbol{\tau}_T^g) p(\boldsymbol{\tau}_T^x) \prod_{t=1}^{T} p_{\theta^g}(\boldsymbol{\tau}_{t-1}^g | \boldsymbol{\tau}_t^g) p_{\theta^x}(\boldsymbol{\tau}_{t-1}^x | \boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x) p_\phi(\mathcal{O}_{1:N}=1 | \boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_{t-1}^x)
\end{aligned}$$

$$p_{\theta^g}\left(\boldsymbol{\tau}_{t-1}^g \mid \boldsymbol{\tau}_t^g, \mathcal{O}_{1:N}=1\right) \approx \mathcal{N}\left(\boldsymbol{\tau}_{t-1}^g ; \mu_{\theta^g}\left(\boldsymbol{\tau}_t^g, t\right)+\lambda^g \Sigma^t \mathcal{J}\left(\mu_{\theta^g}, \mu_{\theta^x}\right), \Sigma^t\right) \tag{26}$$

$$p_{\theta^x}\left(\boldsymbol{\tau}_{t-1}^x \mid \boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x, \mathcal{O}_{1:N}=1\right) \approx \mathcal{N}\left(\boldsymbol{\tau}_{t-1}^x ; \mu_{\theta^x}\left(\boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x, t\right)+\lambda^x \Sigma^t \mathcal{J}\left(\mu_{\theta^g}, \mu_{\theta^x}\right), \Sigma^t\right) \tag{27}$$

$$\mathcal{J}\left(\mu_{\theta^g}, \mu_{\theta^x}\right)=\nabla_{\mu_{\theta^g}} \log p_\phi(\mathcal{O}_{1:N}=1 \mid \boldsymbol{\tau}^g, \boldsymbol{\tau}^x) \mid_{\substack{\boldsymbol{\tau}^g=\mu_{\theta^g} \\ \boldsymbol{\tau}^x=\mu_{\theta^x}}} \tag{28}$$

λ^g and λ^x are the scaling factors of classifier guidance. One classifier simultaneously guides two diffusion processes, where LL trajectories control the HL goals.

D.3 Asynchronous parallel generation

CHD inherits the conditional generation framework from JDM, so $\boldsymbol{\tau}_t^x$ conditions on τ_t^g, making the parallel generation at the same time t impossible. Nevertheless, we can build an asynchronous structure by rewriting the reverse process as,

$$\begin{aligned}
&p\left(\boldsymbol{\tau}_{0:T}^g, \boldsymbol{\tau}_{0:T}^x \mid \mathcal{O}_{1:N}=1\right) \propto p(\boldsymbol{\tau}_T^g) p(\boldsymbol{\tau}_T^x) \underbrace{p_{\theta^g}(\boldsymbol{\tau}_{T-1}^g | \boldsymbol{\tau}_T^g)}_{\mathcal{P}_T^g} \\
&\cdot \prod_{t=1}^{T-1} \Bigg[\underbrace{p_{\theta^g}(\boldsymbol{\tau}_{t-1}^g | \boldsymbol{\tau}_t^g) p_{\theta^x}(\boldsymbol{\tau}_t^x | \boldsymbol{\tau}_t^g, \boldsymbol{\tau}_{t+1}^x) p_\phi(\mathcal{O}_{1:N}=1 | \boldsymbol{\tau}_t^g, \boldsymbol{\tau}_t^x)}_{\mathcal{P}_{t-1}^{g,x}} \Bigg] \underbrace{p_{\theta^x}(\boldsymbol{\tau}_0^x | \boldsymbol{\tau}_0^g, \boldsymbol{\tau}_1^x) p_\phi(\mathcal{O}_{1:N}=1 | \boldsymbol{\tau}_0^g, \boldsymbol{\tau}_0^x)}_{\mathcal{P}_0^x}
\end{aligned}$$

Then, the new paired variables are $(\boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x)$. More details are introduced in section 3.2, while one important change is the asynchronous classifier guidance on HL diffusion.

$$p_{\theta^g}(\boldsymbol{\tau}_{t-1}^g | \boldsymbol{\tau}_t^g) p_\phi(\mathcal{O}_{1:N}=1 | \boldsymbol{\tau}_t^g, \boldsymbol{\tau}_t^x)=p_{\theta^g}\left(\boldsymbol{\tau}_{t-1}^g \mid \boldsymbol{\tau}_t^g, \mathcal{O}_{1:N}=1\right) \tag{29}$$

As in Eqn. (29), the reverse model generates $\boldsymbol{\tau}_{t-1}^g$, while the classifier can only guide $\boldsymbol{\tau}_t^g$, which makes a mismatch of classifier guidance. To solve this problem, we apply the chain rule of gradient to derive,

$$p_{\theta^g}\left(\boldsymbol{\tau}_{t-1}^g \mid \boldsymbol{\tau}_t^g, \mathcal{O}_{1:N}=1\right) \approx \mathcal{N}\left(\boldsymbol{\tau}_{t-1}^g ; \mu_{\theta^g}\left(\boldsymbol{\tau}_t^g, t\right)+\lambda^g \Sigma^t \mathcal{J}^{\text{Asy}}\left(\boldsymbol{\tau}_t^g, \mu_{\theta^x}\right), \Sigma^t\right) \tag{30}$$

$$\begin{aligned}
\mathcal{J}^{\text{Asy}}\left(\boldsymbol{\tau}_t^g, \mu_{\theta^x}\right) &= \nabla_{\mu_{\theta^g}} \log p_\phi(\mathcal{O}_{1:N}=1 \mid \boldsymbol{\tau}^g, \boldsymbol{\tau}^x) \mid_{\substack{\boldsymbol{\tau}^g=\boldsymbol{\tau}_t^g \\ \boldsymbol{\tau}^x=\mu_{\theta^x}}} \\
&= \nabla_{\boldsymbol{\tau}^g} \log p_\phi(\mathcal{O}_{1:N}=1 \mid \boldsymbol{\tau}^g, \boldsymbol{\tau}^x) \mid_{\substack{\boldsymbol{\tau}^g=\boldsymbol{\tau}_t^g \\ \boldsymbol{\tau}^x=\mu_{\theta^x}}} \frac{\partial \boldsymbol{\tau}_t^g}{\partial \mu_{\theta^g}} \\
&= \sqrt{1-\beta_t} \nabla_{\boldsymbol{\tau}^g} \log p_\phi(\mathcal{O}_{1:N}=1 \mid \boldsymbol{\tau}^g, \boldsymbol{\tau}^x) \mid_{\substack{\boldsymbol{\tau}^g=\boldsymbol{\tau}_t^g \\ \boldsymbol{\tau}^x=\mu_{\theta^x}}}
\end{aligned} \tag{31}$$

The partial derivative $\frac{\partial \boldsymbol{\tau}_t^g}{\partial \mu_{\theta g}} = \sqrt{1-\beta_t}$ represents the derivative of the forward model $q(\boldsymbol{\tau}_t^g \mid \boldsymbol{\tau}_{t-1}^g)$. Therefore, we can easily use the classifier with $\boldsymbol{\tau}_t^g$ to guide the generated $\boldsymbol{\tau}_{t-1}^g$.

D.4 Segment-wise generation

In hierarchical planning problems, the primary function is to reduce the trajectory horizon for lower data complexity. A common approach is dividing the trajectory into sub-segments and corresponding subgoals [9]. In the CHD formulation, we apply the segment-wise generation for both LL planner (Eqn. (12)) and hierarchical classifier (Eqn. (13)). In the following, we show the changed loss functions for segment-wise generation.

$$p_{\theta^x}\left(\boldsymbol{\tau}^x_{t-1,1:N} \mid \boldsymbol{\tau}^g_{t-1}, \boldsymbol{\tau}^x_{t,1:N}\right) = \prod_{i=1}^{N} p_{\theta^x}\left(\boldsymbol{\tau}^x_{t-1,i} \mid g_{t-1,i}, \boldsymbol{\tau}^x_{t,i}\right)$$

$$\begin{aligned}\mathcal{L}_{\text{Diff}} &= \mathbb{E}_{\boldsymbol{\tau}_0^g, \boldsymbol{\tau}_0^x, \epsilon^g, \epsilon^x, t}\left[\left\|\epsilon^g - \epsilon^g_{\theta^g}(\boldsymbol{\tau}^g_t, t)\right\| + \left\|\epsilon^x - \epsilon^x_{\theta^x}(\boldsymbol{\tau}^g_{t-1}, \boldsymbol{\tau}^x_t, t)\right\|\right] \\ &= \mathbb{E}_{\boldsymbol{\tau}_0^g, \boldsymbol{\tau}_0^x, \epsilon^g, \epsilon^x, t}\left[\left\|\epsilon^g - \epsilon^g_{\theta^g}(\boldsymbol{\tau}^g_t, t)\right\| + \sum_{i=1}^{N}\left\|\epsilon^x_i - \epsilon^x_{\theta^x}(g_{t-1,i}, \boldsymbol{\tau}^x_{t,i}, t)\right\|\right]\end{aligned} \tag{32}$$

Eqn. (32) is the training loss function with parameterized diffusion models with DDPM [1]. Due to the segmentation, $\epsilon^x = \{\epsilon^x_i\}_{i=1}^N$ is the segment-wise noises, and $\epsilon^x_{\theta^x}(g_{t-1,i}, \boldsymbol{\tau}^x_{t,i}, t)$ is the corresponding models for each segment.

Similarly, we use the segment-wise classifier to optimize the hierarchical model and training losses are as follows,

$$p\left(\mathcal{O}_{1:N} = 1 \mid \boldsymbol{\tau}^g_t, \boldsymbol{\tau}^x_{t,1:N}\right) = \prod_{i=1}^{N} p\left(\mathcal{O}_i = 1 \mid g_{t,i}, \boldsymbol{\tau}^x_{t,i}\right) = \prod_{i=1}^{N} \exp\left(\sum_{k=(i-1)h}^{ih-1} r(s_k, a_k)\right)$$

$$\mathcal{L}_{\text{Classifier}} = \mathbb{E}_{\boldsymbol{\tau}_0^g, \boldsymbol{\tau}_0^x, t}\left[\sum_{i=1}^{N}\left\|p_\phi(g_{t-1,i}, \boldsymbol{\tau}^x_{t,i}, t) - \sum_{k=(i-1)h}^{ih-1} r(s_k, a_k)\right\|\right] \tag{33}$$

Eqn. (33) denotes that the classifier predicts the summation of reward in each segment. In the trajectory planning problems, the rewards are calculated by the spent time steps to achieve the subgoal. For instance, in the maze navigation tasks, the agent receives "+1" for each step upon reaching the subgoal.

D.5 Algorithm

We summarize the CHD's training and sampling algorithms as in Algorithm 1 and 2.

Algorithm 1 CHD Training

```
repeat
    Sample $\boldsymbol{\tau}_0^g, \boldsymbol{\tau}_0^x \sim q(\boldsymbol{\tau}_0^g, \boldsymbol{\tau}_0^x)$
    Sample $t \sim$ Uniform($\{1, \ldots, T\}$)
    Sample $\epsilon^g \sim \mathcal{N}(\mathbf{0}, \mathbf{I}^g)$, $\epsilon^x \sim \mathcal{N}(\mathbf{0}, \mathbf{I}^x)$
    Calculate $\boldsymbol{\tau}_t^g$ and $\boldsymbol{\tau}_t^x$ by forward model $q(\tau_t^\diamond|\tau_0^\diamond) = \mathcal{N}(\tau_t^\diamond; \sqrt{\bar{\alpha}_t}\tau_0^\diamond, (1-\bar{\alpha}_t)\boldsymbol{I}^\diamond), \diamond = \{g, x\}$
    Train diffusion models $\nabla_{\theta^g,\theta^x}\mathcal{L}_{\text{diff}}$ in Eqn. (32)
    Train classifier for rewards $\nabla_\phi \mathcal{L}_{\text{Classifier}}$ with Eqn. (33)
until converged
```

Algorithm 2 CHD Sampling

```
$p(\boldsymbol{\tau}_T^g) \sim \mathcal{N}(\boldsymbol{\tau}_T^g; \mathbf{0}, \mathbf{I}^x)$, $p(\boldsymbol{\tau}_T^x) \sim \mathcal{N}(\boldsymbol{\tau}_T^x; \mathbf{0}, \mathbf{I}^g)$
Calculate $\boldsymbol{\tau}_{T-1}^g \sim p_{\theta^g}(\boldsymbol{\tau}_{T-1}^g | \boldsymbol{\tau}_T^g)$
for $t = T-1, \dots, 1$ do
    Calculate reverse processes $\boldsymbol{\tau}_{t-1}^g \sim p_{\theta^g}(\boldsymbol{\tau}_{t-1}^g | \boldsymbol{\tau}_t^g)$ and $\boldsymbol{\tau}_t^x \sim p_{\theta^x}(\boldsymbol{\tau}_t^x | \boldsymbol{\tau}_t^g, \boldsymbol{\tau}_{t+1}^x)$ in parallel
    Calculate $p_\phi(\mathcal{O}_{1:N} = 1 | \boldsymbol{\tau}_t^g, \boldsymbol{\tau}_t^x)$ and apply classifier guidance by Eqn. (30) and (27)
end for
Calculate $p_{\theta^x}(\boldsymbol{\tau}_0^x | \boldsymbol{\tau}_0^g, \boldsymbol{\tau}_1^x)$ and apply LL classifier guidance $p_\phi(\mathcal{O}_{1:N} = 1 | \boldsymbol{\tau}_0^g, \boldsymbol{\tau}_0^x)$
Return $\boldsymbol{\tau}_0^g$ and $\boldsymbol{\tau}_0^x$
```

D.6 Can JDM adapt to a hierarchical planner?

The joint diffusion model (JDM) is equivalent to one diffusion model with two variables, so it naturally satisfies the property 1. We simplify JDM and utilize three structures to achieve all 3 properties. This raises a question: Can we directly adapt JDM to achieve properties 2 and 3 without any simplification? Can we add parallel generation and segment-wise generation to JDM? The general answer is **No**.

As we stated in the CHD derivation, at time t, $\boldsymbol{\tau}_t^x$ is conditioned on $\boldsymbol{\tau}_t^g$, which forbids synchronous parallel generation. Therefore, we change the reverse step sequence to be $(\boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x)$. However, for JDM the $\boldsymbol{\tau}_{t-1}^g$ also conditions on $\boldsymbol{\tau}_t^x$, meaning the inter-relation cannot be simply broken via asynchronous generation. And this triggers the idea of simplifying JDM to break the chain.

In addition, applying segment-wise generation to JDM cannot efficiently reduce the trajectory horizon. Similarly to CHD, we can divide the trajectory into sub-segments and the corresponding subgoals. This segmentation can be applied to LL planner and classifier, but cannot work on HL planner. Since the HL segment-wise generation becomes $p_{\theta^g}(g_{t-1,i} \mid g_{t,i}, \boldsymbol{\tau}_{t,i}^x)$, which only focuses on generating one subgoal and neglecting the planning of coarse, long-horizon subgoals for the whole trajectory. Therefore, the HL planner cannot adopt segment-wise generation and conditions with the whole long-horizon LL trajectory. On the contrary, CHD breaks the correlation of HL planner and LL trajectory in the diffusion model and utilizes a segment-wise classifier instead.

In short, JDM is not naturally suitable for hierarchical planning. CHD adopts a simplification of JDM and modules for the hierarchical planner. Nevertheless, it is likely to have other methods to build hierarchical planners that also achieve 3 properties.

D.7 Advantages of CHD over BHD

In this section, we claim two advantages of CHD over the BHD. They can theoretically support and validate the enhanced performances of CHD in the long-horizon planning tasks.

1. CHD better approximates the JDM.

Comparing the variational bounds in (21), Eqn. (17) and (22). We can regard both BHD and CHD as a simplifications of the JDM. However, we can prove that CHD's is "closer" to JDM than BHD in terms of the KL divergence. We first denote the Kullback–Leibler(KL)-divergence of CHD and BHD to JDM.

$$D_{\mathrm{KL}}(L^{\mathrm{JDM}} \| L^{\mathrm{BHD}}) = \mathbb{E}_q \left[\sum_{t=1}^{T} \log \frac{p_{\theta^g}(\boldsymbol{\tau}_{t-1}^g | \boldsymbol{\tau}_t^g, \boldsymbol{\tau}_t^x)}{p_{\theta^g}(\boldsymbol{\tau}_{t-1}^g | \boldsymbol{\tau}_t^g)} + \log \frac{p_{\theta^x}(\boldsymbol{\tau}_{t-1}^x | \boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x)}{p_{\theta^x}(\boldsymbol{\tau}_{t-1}^x | \boldsymbol{\tau}_0^g, \boldsymbol{\tau}_t^x)} \right] \tag{34}$$

$$D_{\mathrm{KL}}(L^{\mathrm{JDM}} \| L^{\mathrm{CHD}}) = \mathbb{E}_q \left[\sum_{t=1}^{T} \log \frac{p_{\theta^g}(\boldsymbol{\tau}_{t-1}^g | \boldsymbol{\tau}_t^g, \boldsymbol{\tau}_t^x)}{p_{\theta^g}(\boldsymbol{\tau}_{t-1}^g | \boldsymbol{\tau}_t^g)} \right] \tag{35}$$

$$D_{\mathrm{KL}}(L^{\mathrm{JDM}} \| L^{\mathrm{BHD}}) - D_{\mathrm{KL}}(L^{\mathrm{JDM}} \| L^{\mathrm{CHD}}) = \mathbb{E}_q \left[\sum_{t=1}^{T} \log \frac{p_{\theta^x}(\boldsymbol{\tau}_{t-1}^x | \boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x)}{p_{\theta^x}(\boldsymbol{\tau}_{t-1}^x | \boldsymbol{\tau}_0^g, \boldsymbol{\tau}_t^x)} \right] \tag{36}$$

According to the imitation learning from demonstration, we know that

$$p_{\theta^x}(\boldsymbol{\tau}_{t-1}^x | \boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x) \geq p_{\theta^x}(\boldsymbol{\tau}_{t-1}^x | \boldsymbol{\tau}_0^g, \boldsymbol{\tau}_t^x),$$

since $p_{\theta^x}(\boldsymbol{\tau}_{t-1}^x | \boldsymbol{\tau}_{t-1}^g, \boldsymbol{\tau}_t^x)$ is the correct diffusion process and $p_{\theta^x}(\boldsymbol{\tau}_{t-1}^x | \boldsymbol{\tau}_0^g, \boldsymbol{\tau}_t^x)$ is only a special case. Finally, we can validate that $D_{\mathrm{KL}}(L^{\mathrm{JDM}} \| L^{\mathrm{BHD}}) \geq D_{\mathrm{KL}}(L^{\mathrm{JDM}} \| L^{\mathrm{CHD}})$, the CHD's variational bound is closer to JDM than BHD.

2. CHD achieves higher trajectory optimality than BHD.

We claim that CHD achieves a higher probability of maximum reward optimality than BHD. We first compare the formulation of trajectory optimization:

$$\text{BHD:} \quad p(\boldsymbol{\tau}^x, \boldsymbol{\tau}^g \mid O = 1) = p(\boldsymbol{\tau}^g) p_{\theta^x}(\boldsymbol{\tau}^x \mid \boldsymbol{\tau}^g) p(O = 1 \mid \boldsymbol{\tau}^g) \tag{37}$$

$$\text{CHD:} \quad p(\boldsymbol{\tau}^x, \boldsymbol{\tau}^g \mid O = 1) = p(\boldsymbol{\tau}^g) p(\boldsymbol{\tau}^x \mid \boldsymbol{\tau}^g) p(O = 1 \mid \boldsymbol{\tau}^x, \boldsymbol{\tau}^g) \tag{38}$$

Therefore, the claim can be verified by proving CHD's classifier achieves a higher probability of optimization than BHD $p(O = 1 \mid \boldsymbol{\tau}^g) \leq p(O = 1 \mid \boldsymbol{\tau}^x, \boldsymbol{\tau}^g)$. We define arbitrary LL trajectories $\tilde{\boldsymbol{\tau}}^x$, and the conditional optimality probability can be marginalized as:

$$p(O = 1 \mid \boldsymbol{\tau}^g) = \int p(O = 1 \mid \tilde{\boldsymbol{\tau}}^x, \boldsymbol{\tau}^g) p(\tilde{\boldsymbol{\tau}}^x \mid \boldsymbol{\tau}^g) d\boldsymbol{\tau}^x \tag{39}$$

We also know that the optimality (maximum reward) is determined by the true $\boldsymbol{\tau}^x$ sampled from the demonstration so that the optimality condition on arbitrary $\tilde{\boldsymbol{\tau}}^x$ is always less than the true $\boldsymbol{\tau}^x$.

$$p(O = 1 \mid \tilde{\boldsymbol{\tau}}^x, \boldsymbol{\tau}^g) \leq p(O = 1 \mid \boldsymbol{\tau}^x, \boldsymbol{\tau}^g) \tag{40}$$

Therefore, we marginalize both sides:

$$\int p(O = 1 \mid \tilde{\boldsymbol{\tau}}^x, \boldsymbol{\tau}^g) p(\tilde{\boldsymbol{\tau}}^x \mid \boldsymbol{\tau}^g) d\tilde{\boldsymbol{\tau}}^x \leq \int p(O = 1 \mid \boldsymbol{\tau}^x, \boldsymbol{\tau}^g) p(\tilde{\boldsymbol{\tau}}^x \mid \boldsymbol{\tau}^g) d\tilde{\boldsymbol{\tau}}^x \tag{41}$$

Also, we know the integration of a probability is smaller than 1 $\int p(\tilde{\boldsymbol{\tau}}^x \mid \boldsymbol{\tau}^g) d\tilde{\boldsymbol{\tau}}^x \leq 1$. Bring this to the inequality and we finally prove that CHD achieves higher optimality than BHD. $p(O = 1 \mid \boldsymbol{\tau}^g) \leq p(O = 1 \mid \boldsymbol{\tau}^x, \boldsymbol{\tau}^g)$. Equality holds when the optimality O is independent from HL subgoals $\boldsymbol{\tau}^x$.

E Experiment Details

E.1 Maze Navigation Experiments

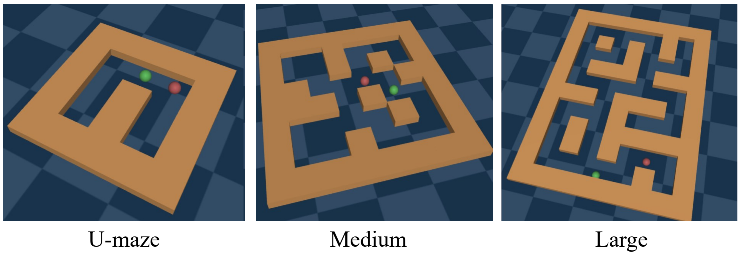


Figure 8: Maze navigation environment in D4RL. The agent start position (greed dot) is randomly chosen in the maze, and the goal position (red dot) is either fixed (Maze2D) or also randomized (Multi2D). The agent will get a “+1” reward when reaching the goal.

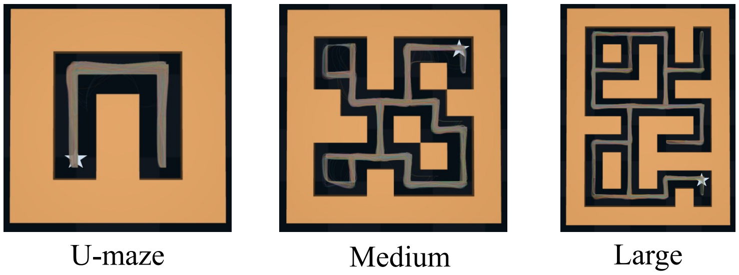


Figure 9: Demonstration in maze navigation environment. In the D4RL dataset, trajectories consist of states and actions. All trajectories do not have pre-defined start or goal position, therefore the trajectory segments may be redundant and sub-optimal. In addition, the agent can only observe the states rather than wall position.

Environments and Datasets

Maze navigation is a long-horizon trajectory planning environment in the D4RL dataset (Maze2D) [12]. The environment has three maze configurations with different sizes: U-maze, Medium and Large. In the simulator, the agent is initially placed in the maze and the target is to plan a feasible trajectory to reach the goal position. For each maze configuration, D4RL can either set a fixed goal position called “Maze2D” or a random goal position in the maze called “Multi2D”, increasing the tasks’ variance and requiring higher generalization ability (Fig. 8). In addition, the agent gets a “+1” reward when it reaches the vicinity of the goal position. So in a maze navigation environment, the planned trajectory should not only successfully reach the goal, but also reduce the time spent traveling to earn higher cumulated rewards.

In maze navigation, the agent is represented by a 2D point that moves in a closed maze. The state is $s = (x, y, v_x, v_y)$ and the action is $a = (v_x, v_y)$. D4RL provides offline demonstrations of the agent traveling in the maze. However, the trajectories do not have definite start or goal positions so they are sub-optimal for the goal-reaching problems (Fig. 9). In addition, since the demonstrations only provide state and action, the agent is unaware of the walls and obstacles in the maze, so the planners can only distill the feasible paths from the demonstrations rather than the wall position.

We finally evaluate the goal-reaching performance using the default method in D4RL. When the agent reaches the goal, it gets a “+1” reward and we calculate the overall reward in one episode. Then we applied maximin normalization and $\times 100$ to compare the results. Higher values mean that the trajectory used fewer steps to reach the goal. We run the experiments with 150 different seeds and report and mean and standard deviation of the results.

Baselines

We compare our CHD method with prior diffusion planner methods.

- **Diffuser** [4] is the initial diffusion model applied in trajectory planning. It leverages the diffusion model to predict both states and actions in the future. To enable goal-conditioned generation and maximum-reward trajectory optimization, Diffuser adopted endpoint inpainting [62] method and classifier guidance [63].
- **Decision diffuser** (DD) [5] improved Diffuser with diffusion transformer backbone and applied classifier-free guidance for trajectory optimization.
- **BHD** (Appendix B) is the general baseline hierarchical diffuser method. It sequentially plans the HL subgoals and then uses subgoals as the endpoint condition to generate LL trajectory segments. The HL planner is guided by the maximum-reward classifier.
- **HDMI** [11] is an improved instance of BHD. The difference is that HDMI pre-processes the dataset using graph search to generate better subgoals. Therefore, it performs better than BHD.
- **SHD** [9] is a simple implementation of BHD. It evenly segments the trajectories and uses endpoints are subgoals. With careful design, SHD further outperforms HDMI.

Since DD, SHD and HDMI did not provide the code for maze navigation, we directly consulted the results reported in their paper for comparison.

Implementation

Our implementation was built on the code released in Diffuser (code here). We first reproduced the Diffuser code and constructed the BHD and CHD algorithms. To ensure a fair comparison, we used U-net as the diffusion backbone and classifier guidance for reward maximization. The inpainting method was adopted for the endpoint-constraint of start and goal positions.

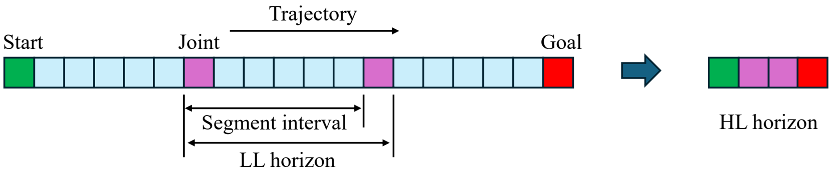


Figure 10: Trajectory segmentation for hierarchical diffuser training.

In the **training** stage (Fig. 10), we sampled fix-length trajectories from the offline demonstrations. Then we evenly divided the trajectories into sub-segments. The HL subgoals consist of the start, intermediate joint points, and the goal point, while LL trajectories are the segments. The segment-wise rewards are the "negative number of steps before reaching the goal" in the segment, which was identical to HDMI [11]. This design can encourage the planner to "speed up" the agent and reduce traveling time. We applied this reward design in the HL classifier in BHD and the coupled hierarchical classifier in CHD.

In the **sampling** stage, we first initialize the agent and simulator to observe the start and goal positions. Then we applied the method to predict the further trajectory in a finite horizon. Note that the planning was open-loop and only executed at the initial state. In the downstream rollouts, a PD controller (implemented in Diffuser) was employed to track the planned trajectory. Finally, the simulator reported the overall rewards, namely the steps that the agent reaches the goal. Due to the different horizon lengths and maze configuration, D4RL normalized the overall reward with the distribution in the demonstrations. We followed the tradition and reported the experimental results in Table 3.

We present the **hyperparameters** used in our implementation in Table 5. For Diffuser, we directly adopted the hyperparameters from the official code. Then, we adapted the hyperparameters in BHD and CHD to achieve a hierarchical structure. A notable modification was the reduction of diffusion steps, as the data distribution for shorter horizons is less complex and therefore easier to model.

	Episode length	Segment[2] interval	Planner horizon			Diffusion steps		
			Single[1]	HL[2]	LL[2]	Single	HL	LL
U-maze	300	31	120	5	32	64	32	32
Medium	600	31	320	12	32	256	32	32
Large	800	31	448	16	32	256	32	32

Table 5: Hyperparameters in the maze navigation experiments. [1] Single means the single-layer Diffuser. [2] In both BHD and CHD, the demonstration trajectories are segmented with the interval. HL subgoals are continuous joint endpoints; LL segments are the divided trajectories.

Additionally, the segment interval is defined as LL horizon $- 1$, since the endpoints correspond to two adjacent LL segments.

We **visualize** additional results in the Large Maze2D environment in Fig. 11 (on the next page). In these visualizations, the goal position is fixed at the bottom-right corner, while the start position varies. Each row in the figure corresponds to a specific start position. The tasks can be categorized into three main scenarios:

1. **Start and goal are very far apart.** The primary challenge in this scenario is to plan feasible, long-horizon, collision-free trajectories. Representative examples include $(1, 4)$, $(1, 8)$, $(2, 4)$, $(2, 6)$, $(3, 5)$, and $(7, 1)$. In general, the Diffuser can generate paths connecting the start and goal positions; however, it struggles with local control, resulting in collisions in cases like $(1, 4)$ and $(3, 5)$. BHD improves planning in cases like $(2, 6)$ by incorporating long-horizon considerations in the HL. However, due to classifier guidance being applied only at the HL, it sometimes produces unachievable subgoals, as seen in $(1, 8)$ and $(3, 5)$. CHD demonstrates superior performance in most cases, producing near-optimal paths in $(1, 4)$, $(1, 8)$, and $(3, 5)$, although it also exhibits undesirable results, such as in $(2, 4)$. In the special case of $(7, 1)$, where the planning horizon is insufficient to reach the goal, CHD shows an attempt to generate **smoother** paths through its optimization process.
2. **Start and goal are near, with multi-modal paths.** In these cases, the agent can initially move in multiple directions, with more than one valid path to the goal. Examples include $(3, 6)$, $(3, 9)$, and $(5, 10)$. Here, the Diffuser performs poorly, generating unreasonable paths. BHD produces more reasonable paths but with noticeable local flaws. CHD significantly enhances performance, successfully generating feasible paths to the goal. However, some paths, such as $(5, 10)$, still exhibit redundant turns.
3. **Start and goal are much closer.** Despite the shorter distances, these paths often involve numerous turns at corners, as seen in $(5, 4)$, $(6, 6)$, $(6, 8)$, $(7, 4)$, and $(7, 10)$. In such cases, the planner must optimize trajectories to reduce travel steps while maintaining feasibility. Both the Diffuser and BHD generate generally feasible paths but suffer from redundant turns due to suboptimal demonstrations. The Diffuser's long-horizon trajectory modeling struggles with high variance, making trajectory optimization challenging. BHD, with its coarse HL goals, fails to provide feedback or adjust subgoals effectively. By contrast, CHD leverages coupled diffusion to align HL and LL, enabling superior path optimization that balances feasibility and near-optimality, as observed in $(7, 4)$.

From the experiments and comparisons, we conclude that CHD outperforms both the Diffuser and BHD methods in long-horizon maze navigation tasks. Across different planning scenarios, CHD consistently generates feasible, goal-reaching trajectories by imitating demonstrations and optimizing travel steps through coupled classifier guidance.

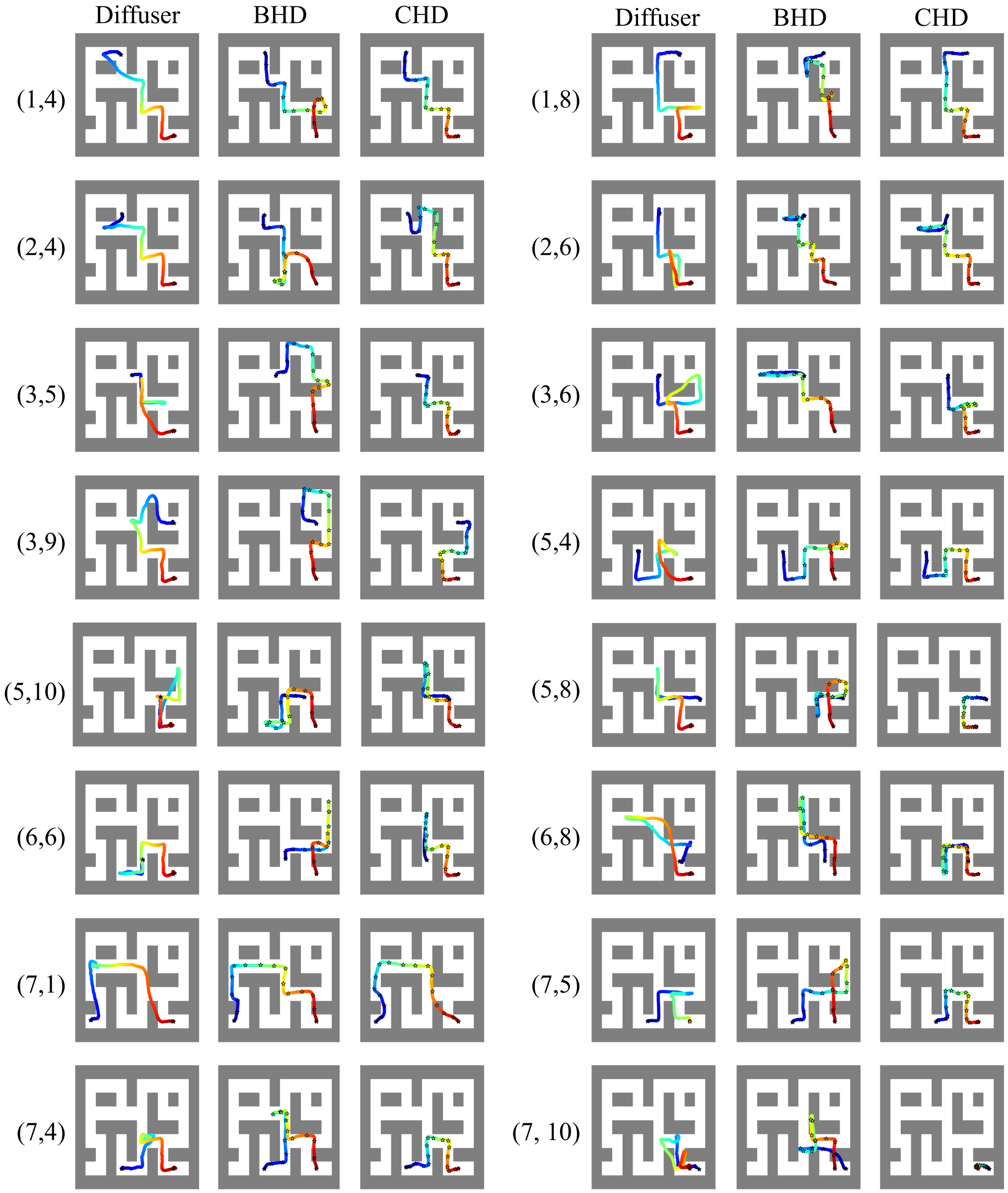


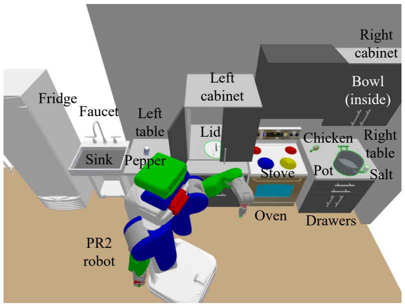
Figure 11: Visualization of maze navigation results in the Maze2D Large environment. The trajectory is from blue start to red goal. The goal position is always on the bottom right, while the start position varies and is marked in each row. ★ represents intermediate sub-goals in BHD and CHD.

E.2 Robot Task Planning Experiments

Environments and Datasets

The robot task planning experiments were conducted using the Kitchen World simulator [15], which is designed for learning-based algorithms in task planning [13, 15] and supports integration with downstream solvers like PDDLStream [64]. In Kitchen World, the PR2 dual-arm robot moves in the kitchen scenario, manipulating rigid and articulated objects to finish certain tasks. In our experiments, we created a customized task-planning benchmark with extended horizons to evaluate performance. During the training phase, we utilized offline demonstrations to train the planners. In the sampling phase, the planner generated sub-goal states and actions, which were subsequently executed using the PDDLStream solver as the policy. In short, the simplified PDDL maps the semantic action and targets to the robot and objects in the kitchen world simulation and plans trajectories to satisfy each primitive motion.

Task planning was formalized in a simplified PDDL space [8]. The state space is represented as $s = [(\texttt{Object}, \texttt{Position})_n]$, where n is the number of objects, and the action space is defined as $a = [(\texttt{Motion}, \texttt{Target})]$. The `Object` set includes all movable rigid and articulated objects in the environment, such as $[\texttt{Bowl}, \texttt{Pot}, \texttt{Cabinet}, \ldots]$. The `Position` specifies the location or state of the object, e.g., $[\texttt{On-table}, \texttt{In-cabinet}, \texttt{Open}, \texttt{In-Bowl}, \ldots]$. The `Motion` component of actions describes the robot's behavior at each step, encompassing $[\texttt{Pick}, \texttt{Place}, \texttt{Grasp}, \texttt{Push}, \texttt{Move}, \ldots]$. The `Target` specifies the intended target of the motion, such as $[\texttt{Chicken}, \texttt{Cabinet}, \texttt{Right-Table}, \ldots]$.



State space:

Objects: [Fridge, Faucet, Pepper, Left-Cabinet, Lid, Stove, Oven, Drawers, Chicken, Pot, Salt, Bowl, Right-Cabinet]

Positions: [In-Fridge, Opening, Closed, On-Pot, In-Left-Cabinet, In-Right-Cabinet, In-Drawers, In-Sink, On-Stove, In-Oven, On-Left-Table, On-Right-Table, In-Pot, In-Bowl,]

Action space:

Motions: [Move, Pick, Place, Grasp, Pull, Push, Turn-on, Turn-off, Sprinkle, Scoop]

Targets: [On-Pot, On-Right-Table, In-Sink, Stove, In-Bowl, In-Left-Cabinet, Left-Table, Fridge, …]

Inputs:

Task: cook the chicken

Initial states:

[(Pot, On-Right-Table),
(Chicken, On-Right-Table),
(Salt, On-Right-Table),
(Pepper, On-Left-Table),
(Lid, In-Left-Cabinet),
(Bowl, In-Right-Cabinet),
(Fridge, Closed),
(Faucet, Closed),
(Stove, Closed),
……
]

Example demonstrations (minimal):

HL states (changed only):

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
(Chick., In-Sink)	(Chick., In-Pot)	(Salt, In-drawer)	(Pepper, In-drawer)	(Pot, In-Sink)	(Pot, On-Stove)	(Lid, On-Pot)	(Bowl, On-RTable)	(Lid, On-LTable)	(Chick., In-Bowl)

LL actions:

1) (Move, RTable)	(Pick, Chick.)	(Pick, Salt)	(Move, LTable)	(Pick, Pot)	(Pick, Pot)	(Move, LCabinet)	(Move, RCabinet)	(Pick, Lid)	(Move, Stove)
2) (Pick, Chick.)	(Move, RTable)	(Sprinkle, In-Pot)	(Pick, Pepper)	(Move, Sink)	(Move, Stove)	(Grasp, LCabinet)	(Grasp, RCabinet)	(Move, LTable)	(Pick, Chick.)
3) (Move, Sink)	(Place, In-Pot)	(Grasp, drawer)	(Move, RTable)	(Place, In-Sink)	(Place, On-stove)	(Pull, LCabinet)	(Pull, RCabinet)	(Place, On-LTable)	(Move, RTable)
4) (Place, In-Sink)		(Pull, drawer)	(Sprinkle, In-Pot)	(Grasp, Faucet)	(Turn-on, Stove)	(Pick, Lid)	(Pick, Bowl)		(Place, In-Bowl)
5) (Grasp, Faucet)		(Place, In-drawer)	(Grasp, drawer)	(Turn-on, Faucet)		(Push, LCabinet)	(Place, On-RTable)		
6) (Turn-on, Faucet)		(Push, drawer)	(Pull, drawer)	(Turn-off, Faucet)		(Move, Stove)	(Push, RCabinet)		
7) (Turn-off, Faucet)			(Place, In-drawer)			(Place, On-Pot)			
8)			(Push, drawer)			(Turn-off, Stove)			

Figure 12: Illustration of task planning in the Kitchen World environment. Objects are randomly initialized in the scene, and a target task is specified. Based on the initial states, the program generates high-level (HL) subgoal states and low-level (LL) actions to complete the task. We use 'Left' and 'Right' as 'L' and 'R' for simplicity.

We constructed the dataset around the task of "cooking meals" within the kitchen environment, comprising 10 sub-tasks. Fig. 12 illustrates an example scene and demonstration. The state and action spaces, as well as initial states, are predefined in the scene. The program then generates HL subgoal states and LL actions required to accomplish the target task. The order of the 10 sub-tasks is interchangeable, provided they collectively achieve the target, which enhances the model's generalization capability in real-world scenarios. Each sub-task has a maximum of 10 actions and will receive a "+1" reward when the sub-task is finished. Therefore, the planner should generate successful subgoals and actions while reducing the number of actions for each sub-task.

To diversify demonstrations, we randomized the initial and terminal states of objects, varied the sub-task sequences, and introduced redundant behaviors. This approach ensured suboptimal demonstrations with extended trajectories. Based on the average trajectory lengths in the demonstrations, tasks were categorized into Easy (50 steps), Medium (70 steps), and Hard (90 steps). Additionally, a subset of tasks with fixed terminal states was defined as "Single," while the full dataset with randomized terminal states was labeled "Multi." The dataset consisted of $10,000$ training samples and $1,000$ validation samples. Finally, we evaluated the planned tasks in the simulator by measuring the number of completed tasks and the steps required to execute them. We evaluate the planners' performance in terms of the number of successfully finished sub-tasks and the normalized steps to finish each sub-task. The preferred optimal planning should complete more sub-tasks and the cumulated action steps should be minimized.

Baselines

We applied baseline methods in the task planning environment. These baselines fall into three categories: auto-regressive transformers, single-layer diffusion models, and hierarchical diffusion models. The methods are:

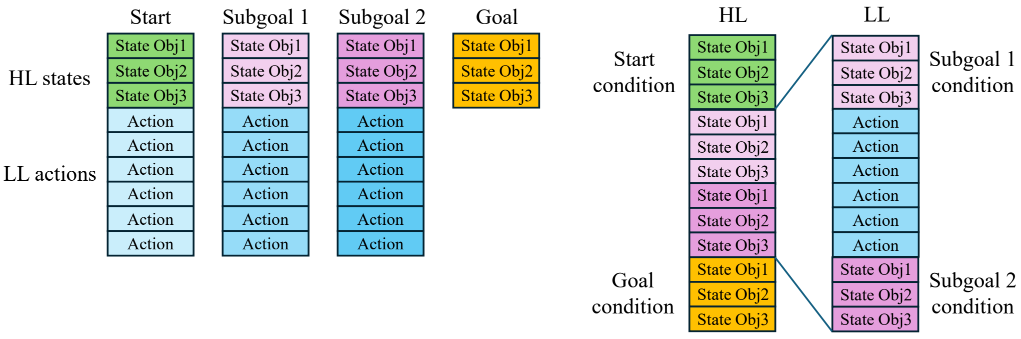
1. **Large-Language Models (LLMs)** [15, 65], such as ChatGPT-4o, are highly effective tools for task planning. They leverage general commonsense knowledge learned from web-scale datasets to break down complex tasks into sub-tasks. In our experiments, we adapted the prompts from [15] and fed the training data to the LLM agent as in-context learning. This allowed the LLM agent to analyze tasks and generalize to new initial and goal states. During the sampling stage, we provided the initial and goal states, and ChatGPT-4o directly output all LL actions.
2. **Transformer** [14] uses an auto-regressive structure to predict the next best token, making it naturally suitable for generating long-horizon trajectories. We employed the transformer as the backbone for task planning. Using the demonstrations, we trained the default "gpt2" transformer network in Huggingface (Transformers). During the sampling stage, the initial and target states were given as input, and a greedy policy was used to predict subsequent tokens until the end-of-sequence (EoS) token was reached.
3. **Diffuser** [4] was originally designed for continuous trajectory planning. To adapt it for planning in discrete token space, we applied the Bits Diffusion method [18] to analogize the tokens. We then trained and validated the Diffuser for task planning, treating the initial and goal states as endpoint conditions.
4. **BHD** (Baseline Hierarchical Diffusion, see Appendix B) is the hierarchical diffusion baseline method. Similar to Diffuser, we applied Bits Diffusion to analogize tokens. We trained both HL and LL diffusers to generate HL subgoal states and LL actions. The HL planner used the initial and goal states as endpoint constraints, while the LL planner used the HL subgoals as endpoint constraints.

Note that the LLMs, Transformer, and Diffuser baselines only generate LL actions, whereas both BHD and our proposed CHD plan generate HL subgoal states and LL actions.

Implementation

We designed a paradigm for task planning with the hierarchical diffusion structure. In Fig. 13, the demonstrations are divided into sub-tasks with their HL subgoal states and LL actions. Then we re-organized the demonstrations to construct the HL and LL data. The HL planner has start and goal conditions, and it generates the intermediate subgoals. The LL planner conditions on two adjacent states and fills the actions that transit between two states. Finally, the actions were executed in the environment with the PDDLStream solver.

Table 6 shows the hyperparameters used in the task planning experiments. In each environment, we set the episode length according to the average horizon of demonstrations. We set 10 HL subgoals for all tasks while the LL horizon is sufficiently longer than the demonstration. Therefore, the LL demonstrations are filled with padding tokens "[PAD]" at the end to satisfy the planning horizon.



(a) Demonstrations

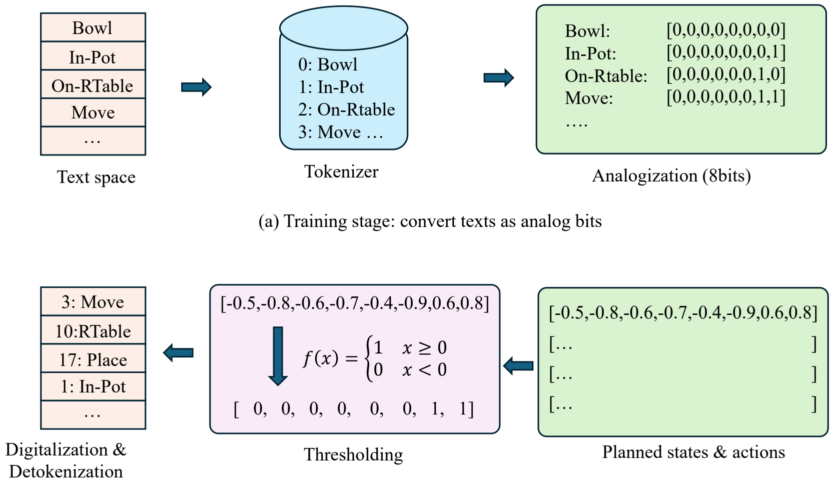
(b) Hierarchical planning

Figure 13: Illustration of demonstrations and hierarchical planning. (a) The demonstrations are divided into several subgoals with HL states and LL actions. (b) The HL and LL planners are trained with the reorganized data format.

	Average demo. length	Episode length	Segment interval	Planner horizon			Diffusion steps		
				Single[1]	HL[2]	LL[2]	Single	HL	LL
Easy	50	100	10	96	10	10	256	32	32
Medium	70	120	10	112	10	12	256	32	32
Hard	90	140	10	128	10	14	256	32	32

Table 6: Hyperparameters in the task planning experiments. [1] Single planners horizon means the single-layer methods like LLMs, Transformers and Diffuser. [2] The HL planner generates 10 subgoal states; while LL planner generates the actions given adjacent states.

Comprising Diffuser and the hierarchical methods BHD and CHD, we also reduced the diffusion steps, since the hierarchical planners have much lower horizon and distribution complexity.



(a) Training stage: convert texts as analog bits

(b) Sampling stage: convert planned values as texts

Figure 14: Illustration of Bits Diffusion for task planning experiments.

We elaborate on implementing Bits Diffusion [18] in the task planning experiments. In Fig. 14, we first defined the whole state and action spaces, which were tokenized using the PreTrainedTokenizerFast in Huggingface (code). Then we converted the tokens as 8 Bits analog values. Since the state and action have two tokens, the diffusion model generates values with dimension “horizon ×16”. In the sampling stage, the planners sampled values in continuous space. And we converted them with thresholding to binary values. Finally, the bits were digitalized and de-tokenized back to the texts for the following steps.

E.3 Real-robot Experiments

Please see the video in the supplementary material or online website for the real-robot experiment results.

Environments and hardware

We conduct our real-robot experiments using the Fetch mobile manipulator in a typical home environment comprising three rooms: the kitchen, living room, and dining room. Fig. 15 presents a top-down view of the experimental setup. The left side represents the **living room**, furnished with a sofa, coffee table, trash bin, chair, and cloth shelf. The **dining room**, located at the bottom right, contains a dining table surrounded by chairs. Items are initially placed on the dining table for subsequent interactions. The **kitchen**, in the top right, is the most critical area, equipped with various appliances and objects, including a washing machine, sink, dishwasher, microwave, fridge, and other tableware.

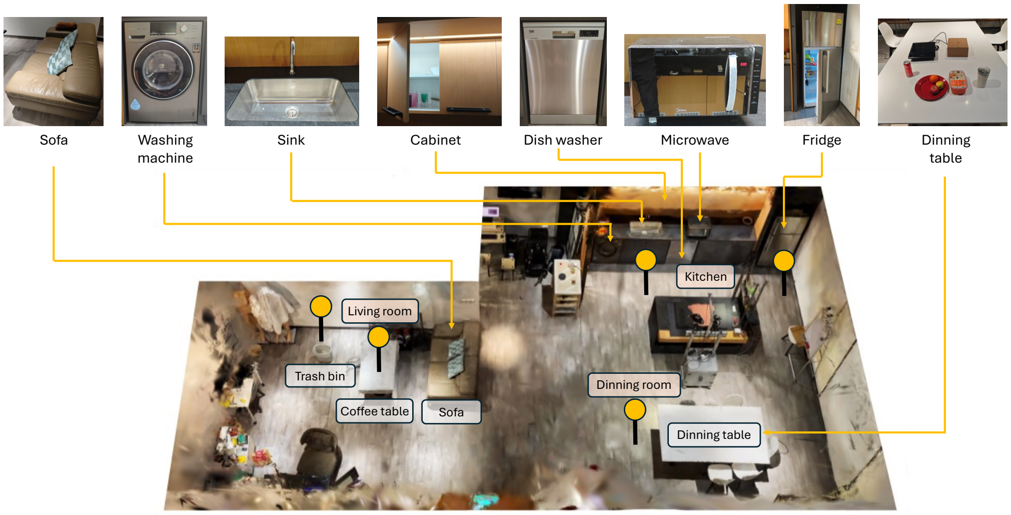


Figure 15: Top-down view of the real-robot experiment scenario.

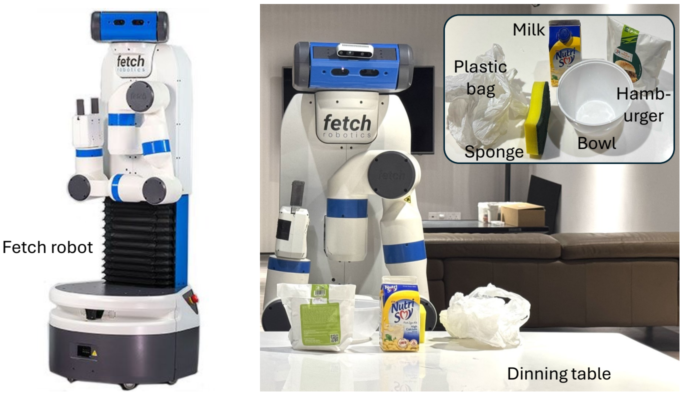


Figure 16: Fetch mobile robot and its environment. Objects are initially placed in clutter on the dining table, and the Fetch robot is tasked with organizing them.

The Fetch robot (Fig. 16) is a mobile manipulator designed for research and industrial automation. Developed by Fetch Robotics, it features a nonholonomic mobile base, a 7-degree-of-freedom (DoF) robotic arm, and a gripper, making it well-suited for tasks such as object manipulation, warehouse logistics, and human-robot interaction. In our experiments, The Fetch robot manipulates various rigid and articulated objects throughout the home.

Task planner implementation

We adopted the same task-planning method described in Appendix E.2. First, we defined the state and action space of the mobile robot. We changed the actions with articulated objects as [`Open`, `Close`]. Next, we manually designed behaviors for randomly organizing items, such as storing objects in containers. Based on these behaviors, we randomly generate a large task-planning dataset by forward sampling, ensuring comprehensive coverage of the possible state-action sequence distribution. We used this dataset to train the CHD. Given any initial state and goal state, the CHD uses inpainting the HL subgoals and LL actions. We show the demonstration of the results in the video.

Task: Prepare a burger for lunch and organize the groceries on the table.

Initial states:

[(Bowl,	On-DinningTable),
(Hamburger,	On-DinningTable),
(Milk,	On-DinningTable),
(PlasticBag,	On-DinningTable),
(Sponge,	On-DinningTable),
(Fridge,	Closed),
(Cabinet,	Closed),
(Microwave,	Closed),
]	

Planned tasks:

HL states:

a) (PlasticBag In-TrashBin)
b) (Bowl, On-CoffeeTable)
(Hamburger, In-Bowl)
c) (Cabinet, Open)
(Sponge, In-Cabinet)
(Cabinet, Closed)
d) (Fridge, Open)
(Milk, In-Fridge)
(Fridge, Closed)

LL actions:

a.1) (Move, DiningTable)
a.2) (Pick, PlasticBag)
a.3) (Move, TrashBin)
a.4) (Place, In-TrashBin)

b.1) (Move, DiningTable)
b.2) (Pick, Bowl)
b.3) (Move, CoffeeTable)
b.4) (Place, On-CoffeeTable)
b.5) (Move, DiningTable)
b.6) (Pick, Hamburger)
b.7) (Move, CoffeeTable)
b.8) (Place, In-Bowl)

c.1) (Move, Cabinet)
c.2) (Open, Cabinet)
c.3) (Move, DiningTable)
c.4) (Pick, Sponge)
c.5) (Move, Cabinet)
c.6) (Place, In-Cabinet)
c.7) (Close, Cabinet)

d.1) (Move, Fridge)
d.2) (Open, Fridge)
d.3) (Move, DiningTable)
d.4) (Pick, Milk)
d.5) (Move, Fridge)
d.6) (Place, In-Fride)
d.7) (Close, Fridge)

Figure 17: Example of real-robot task planning. According to initial states, CHD planned HL subgoal states and LL actions to complete the task.

Fig. 17 shows an example of task planning. Given the initial states of objects, the CHD planner generated HL subgoals states and LL actions. We can categorize them into 4 sub-tasks. a) Drop the plastic bag; b) serve the hamburger; c) put sponge in the cabinet; d) put milk in the fridge. Finally, the robot adopted the real-robot controller to execute the actions to finish all tasks.

Real-robot System Implementation

All the computation for decision-making, perception, and low-level policy is done on a Linux workstation with an NVIDIA RTX 4090 GPU. The initial states are given at the beginning of the tasks for simplification, while the states could easily be constructed by using a VLM given the RGB observation. The goal state, which defines where all objects should be, is generated by the LLM combined with the user's preference, e.g. user specifies the burger served in the bowl on the coffee table. The robot is equipped with several navigation and manipulation skills: `pick, place, move, open, close`. These skills are implemented following the design of our previous mobile manipulation system [21].

The `pick` policy processes text queries in the format `pick(text)`. To identify and segment the target object, we leverage the pre-trained open-vocabulary detection model OWLv2 [66] in combination with the Segment Anything model [67]. The segmented object mask is then used in conjunction with the pre-trained grasping model Contact-GraspNet [68] to determine viable grasping poses. We refine these poses based on orientation constraints and select the one with the highest confidence score. A simple pre-grasp and grasp strategy is applied, with arm trajectories generated using MoveIT's motion planning tools.

The `place` policy operates similarly to the `pick` and also supports text-based queries. Once the segmented point clouds of the target placement area are obtained, the placement position is computed: the center of the object is determined in the X-Y plane, while the height is set by adding 0.15 meters to the highest point in the segmented point clouds. For large, fixed objects or locations such as tables, counters, and trash bins, we simplify the process by using predefined placement locations. Additionally, for challenging placements (e.g., placing objects inside cabinets or fridges), we incorporate imitation learning for simplification.

For navigation, the `move` policy operates with predefined waypoints for all known locations. First, we generate an occupancy map using Gmapping and define navigation waypoints for known locations within the map. Path and motion planning are handled using the ROS Navigation Stack, which provides off-the-shelf algorithms for trajectory generation.

The `open` and `close` functions rely on imitation learning to execute complex actions, such as opening and closing a fridge or a cabinet. We collected an average of 50 human-teleoperated demonstrations per action using a VR controller on a real robot. These demonstrations were then used to train an Action Chunking with Transformers (ACT) [22] model. The model takes RGB-D images and the robot arm's joint states as inputs to predict joint angle movement sequences, enabling smooth execution of these tasks.