Compositional Diffusion with Guided search for Long-Horizon Planning

Utkarsh A. Mishra, David He, Yongxin Chen and Danfei Xu
Georgia Institute of Technology
utkarshm.robo@gatech.edu

Abstract

Generative models have emerged as powerful tools for planning, with compositional approaches offering particular promise for modeling long-horizon task distributions by composing together local, modular generative models. This compositional paradigm spans diverse domains, from multi-step manipulation planning to panoramic image synthesis to long video generation. However, compositional generative models face a critical challenge: when local distributions are multimodal, existing composition methods average incompatible modes, producing plans that are neither locally feasible nor globally coherent. We propose Compositional Diffusion with Guided Search (CDGS), which addresses this *mode averaging* problem by embedding search directly within the diffusion denoising process. Our method explores diverse combinations of local modes through population-based sampling, enforces global consistency through iterative resampling between overlapping segments, and prunes infeasible candidates using likelihood-based filtering. CDGS matches oracle performance on seven robot manipulation tasks, outperforming baselines that lack compositionality or require long-horizon training data. The approach generalizes across domains, enabling coherent text-guided panoramic images and long videos through effective local-to-global message passing. More details: https://cdgsearch.github.io/

1 Introduction

Synthesizing coherent long sequences is a crucial and challenging task, requiring reasoning over extended horizons. This task arises naturally in various domains: robotic actions must enable future steps, parts of a panorama must align semantically, and subjects in a video must remain consistent across hundreds of frames.

Recent work leverages generative models to learn long sequence distributions [24, 3], with diffusion models [49, 20] gaining popularity for modeling multi-modal data [8, 19]. However, full-sequence data is expensive to acquire, and monolithic models fail to generalize beyond training horizons [9]. As an alternative, compositional generation effectively combines short-horizon local distributions to sample long-horizon global plans [62, 41, 37]—e.g., chaining skills for task planning, connecting images into panoramas, or stitching clips into videos. While this improves data-efficiency and allows extrapolation beyond training data, it introduces a **critical challenge**: as local plan distributions become highly *multimodal*, the distribution of global plans inherits combinatorial multi-modality. For example, in the robotics

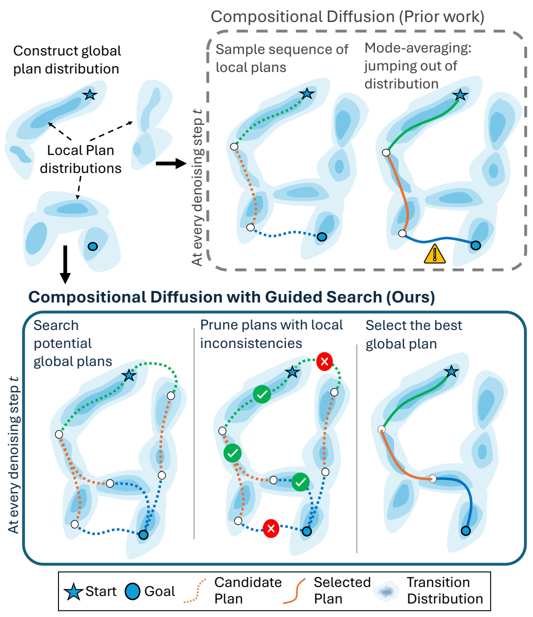

Compositional Diffusion (Prior work)
Construct global plan distribution
Local Plan distributions
Sample sequence of local plans
Mode-averaging: jumping out of distribution
At every denoising step t
Compositional Diffusion with Guided Search (Ours)
Search potential global plans
Prune plans with local inconsistencies
Select the best global plan
At every denoising step t
Start
Goal
Candidate Plan
Selected Plan
Transition Distribution

Figure 1: **Compositional Diffusion with Guided Search (CDGS)** composes short-horizon plan distributions to sample long-horizon goal-directed plans directly at inference. Unlike naïve compositional sampling, it explores diverse plans and filters locally inconsistent paths to avoid "mode averaging", yielding globally coherent plans.

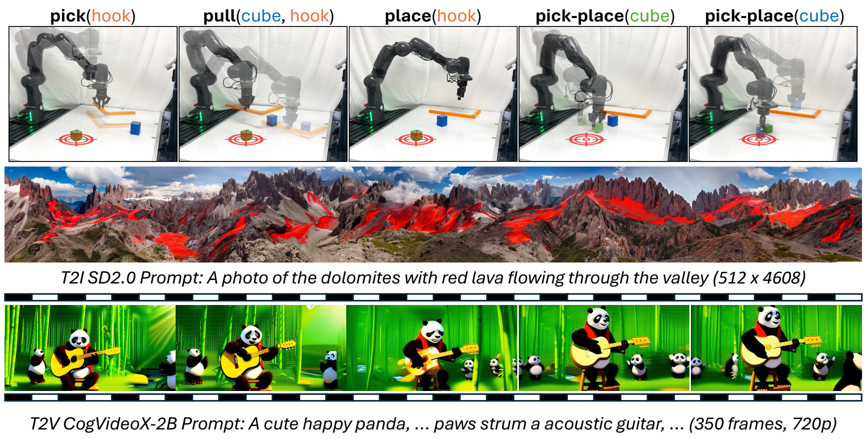

pick(hook)
pull(cube, hook)
place(hook)
pick-place(cube)
pick-place(cube)
T2I SD2.0 Prompt: A photo of the dolomites with red lava flowing through the valley (512 x 4608)
T2V CogVideoX-2B Prompt: A cute happy panda, ... paws strum a acoustic guitar, ... (350 frames, 720p)

Figure 2: **Applications of CDGS.** (Top) Long horizon motion planning: CDGS discovers a valid multi-step plan to move the blue cube to the green cube's original position via : (1) using the hook to pull blue cube in workspace, (2) displace the green cube to make space and (3) moving the blue cube to the target position. (Mid) CDGS generates coherent panoramic images. (Bottom) CDGS can stitch short clips to generate consistent, longer videos.

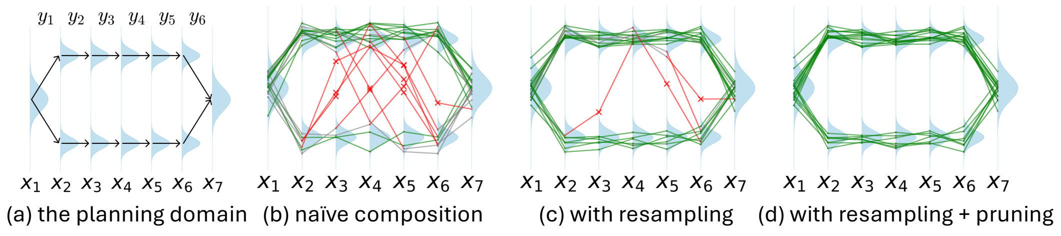
scenario in Fig. 2, because the robot has a large combination of actions and objects it can act on, the search space of possible plans grows exponentially with the length of the planning horizon.

Existing methods for compositional generation offer a promising approach, using *score-averaging* to compose modes of local distributions into a global distribution [62, 41]. However, these methods have an **important limitation**: their inability to handle the combinatorial multi-modality leads them to *average* incompatible local modes (*mode-averaging*), ultimately producing invalid global plans. Addressing such complex multi-modal distributions requires inference methods that jointly reason about compatibility between local modes and effectively navigate the exponentially large search space.

To address the challenge and overcome the limitation, we aim to identify compatible sequences of local modes that compose into a globally coherent plan. Given the diversity and multi-modality of the search space, we take inspiration from classical search techniques and introduce Compositional Diffusion with Guided Search (CDGS), a guided search mechanism integrated into the diffusion denoising process as illustrated in Fig. 1. To facilitate the search during inference, at each diffusion timestep, our method introduces two key components: (i) **iterative resampling** to enhance local-global message passing in compositional diffusion to propose globally plausible candidates, and (ii) **likelihood-based pruning** to remove incoherent candidates that fall into low-likelihood regions due to mode-averaging. Together, these components enable CDGS to efficiently sample coherent long-horizon plans. For robotics tasks, our method outperforms or is on par with baselines that lack compositionality or use long-horizon data for training, respectively. We also show the efficacy of our method in long text-to-image and text-to-video tasks (Fig. 2), producing more coherent and consistent generations.

2 Background

Problem formulation. A long-horizon plan generation problem is characterized by the task of constructing a global plan $\tau = (x_1, \dots, x_N)$ as a sequence of variables x_i, by sampling from the joint distribution $p(\tau)$. The problem becomes goal-directed if τ must connect a given start $x_1 = x_s$ to a desired goal $x_N = x_g$. Such problems arise in diverse domains: long-horizon manipulation planning, panoramic images, and long videos. While modeling the full joint distribution $p(\tau)$ would directly model all dependencies between any x_i, it usually entails end-to-end learning from long-horizon data, which can be infeasible or expensive to obtain. In the absence of long-horizon data, a promising strategy is to approximate the joint distribution $p(\tau)$ with a factor graph of overlapping local distributions that can be learned from short-horizon data. For the joint variable $\tau = (x_1, x_2, \dots, x_N)$, we construct a factor graph [29] connecting variable nodes $\{x_i\}_{i=1}^{N}$ and factor nodes $\{y_j\}_{j=1}^{M}$, where each factor y_j represents the joint distribution of contiguous subsequences of τ. For example, we represent $\tau = (x_1, x_2, \dots, x_5)$ with factors $y_1 = (x_1, x_2, x_3), y_2 = (x_3, x_4, x_5)$. With this, we construct



y_1 y_2 y_3 y_4 y_5 y_6
x_1 x_2 x_3 x_4 x_5 x_6 x_7
(a) the planning domain
(b) naïve composition
(c) with resampling
(d) with resampling + pruning

Figure 3: **Running example.** (a) Consider a 1D-domain of $\{x_{1:7}\}$ variable distributions and $\{y_{1:6}\}$ feasible directed transitions between the variables. There are two feasible long-horizon plans from start (x_1) to goal (x_7): one through the top and one through the bottom. (b) in naive-composition, sampled plans may choose to start in the top and end at the bottom, or vice versa. When this happens, the intermediate models $\{y_{2:5}\}$ will average the modes of intermediate variables $\{x_{2:6}\}$ to satisfy both constraints, manifesting in infeasible transitions (red) (c) adding **iterative resampling** reduces the frequency of mode-averaging (d) adding **pruning** eliminates plans with infeasible y

the joint distribution $p(\tau)$ using the Bethe approximation [60]:

$$p(\tau) := \frac{p(x_1,x_2,x_3)p(x_3,x_4,x_5)\dots}{p(x_3)\dots} = \frac{\prod_{j=1}^{M} p(y_j)}{\prod_{i=1}^{N} p(x_i)^{d_i-1}} \tag{1}$$

where d_i is the degree of the variable node x_i. This representation enables sampling from the long-horizon distribution $p(\tau)$ using only samples drawn from a short-horizon distribution $p(y)$.

Diffusion models. Diffusion models are defined by a forward process that progressively injects noise into the data distribution $p(y^{(0)})$ and a reverse diffusion process that iteratively removes the noise by approximating $\nabla \log p$ to recover the original data distribution. For a given noise injection schedule α_t, forward noising adds a Gaussian noise ε to clean samples s.t. $y^{(t)} = \sqrt{\alpha_t} y^{(0)} + \sqrt{1-\alpha_t}\varepsilon$. With p_t being the distribution of noisy samples, the denoising is performed using the score function $\nabla_{y^{(t)}} \log p_t(y^{(t)})$ often estimated by a neural network $\varepsilon_\theta(y^{(t)},t)$ learned via minimizing the score matching loss [22] given by $\mathbb{E}_{t,y^{(0)}}[|\varepsilon - \varepsilon_\theta(y^{(t)},t)|^2]$. Such a score function allows denoising the noise samples via sampling from

$$p(y^{(t-1)}|y^{(t)}) = \mathcal{N}\left(y^{(t-1)}; \sqrt{\alpha_{t-1}}\hat{y}_0^{(t)} + \sqrt{1-\alpha_{t-1}-\sigma_t^2}\,\varepsilon(y^{(t)},t), \sigma_t^2 \mathbf{I}\right) \tag{2}$$

where $\hat{y}_0^{(t)} = \frac{y^{(t)} - \sqrt{1-\alpha_t}\varepsilon_\theta(y^{(t)},t)}{\sqrt{\alpha_t}}$ is the Tweedie estimate of the clean sample distribution at denoising step t and σ_t controls stochasticity [50]. Several works have leveraged the flexibility of the denoising process in performing post-hoc guidance [19] and plug-and-play generation [34, 11].

Compositional sampling with diffusion models. Under the diffusion model formulation, we can *compositionally* sample [10, 62] from the factor graph representation of $p(\tau)$ by calculating the score $\nabla \log p(\tau)$ as a sum of factor and variable scores following Eq. 1:

$$\nabla \log p(\tau) := \sum_{j=1}^{M} \nabla \log p(y_j) + \sum_{i=1}^{N} (1-d_i)\nabla \log p(x_i) \tag{3}$$

In practice, our factor graph is a chain, so overlapping variables (i.e., the ones shared between neighboring factors y_j and y_{j+1}) have degree $d_i = 2$ while non-overlapping ones have $d_i = 1$ (i.e. their marginals have no contribution to $\nabla \log p(\tau)$). For overlapping variables, we approximate the marginal scores using the average of the conditional score: $\nabla \log p(x_i) \approx \frac{1}{2}\Big[\nabla \log p_{y_j}(x_i|\dots) + \nabla \log p_{y_{j+1}}(x_i|\dots)\Big]$ where $x_i \in y_j \cap y_{j+1}$ denotes the overlapping variable between y_j and y_{j+1}. This, along with the scores of $p(y_j)$ computed from local distribution, allows us to formulate the global compositional score $\nabla \log p(\tau)$ using Eq. 3. While this formulation enables generalization beyond the lengths seen during training, it comes with limitations described in Sec. 3.

3 METHOD

Challenge: Compositional sampling with multi-modal distributions. Solving long-horizon tasks requires constructing a coherent global plan distribution that induces an exponentially large search

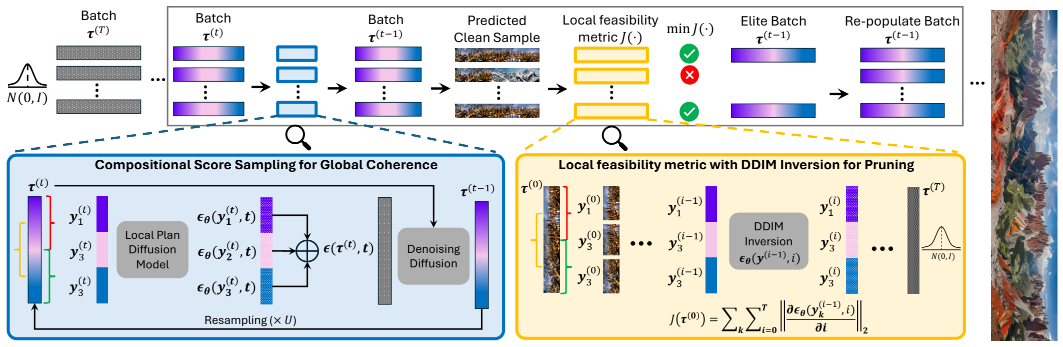

Batch $\tau^{(T)}$
$N(0, I)$
Batch $\tau^{(t)}$
Batch $\tau^{(t-1)}$
Predicted Clean Sample
Local feasibility metric $J(\cdot)$
$\min J(\cdot)$
Elite Batch $\tau^{(t-1)}$
Re-populate Batch $\tau^{(t-1)}$
Compositional Score Sampling for Global Coherence
$\tau^{(t)}$
$y_1^{(t)}$
$y_3^{(t)}$
$y_3^{(t)}$
Local Plan Diffusion Model
$\epsilon_\theta(y_1^{(t)}, t)$
$\epsilon_\theta(y_2^{(t)}, t)$
$\epsilon_\theta(y_3^{(t)}, t)$
$\epsilon(\tau^{(t)}, t)$
Denoising Diffusion
$\tau^{(t-1)}$
Resampling ($\times U$)
Local feasibility metric with DDIM Inversion for Pruning
$\tau^{(0)}$
$y_1^{(0)}$
$y_3^{(0)}$
$y_3^{(0)}$
$y_1^{(i-1)}$
$y_3^{(i-1)}$
$y_3^{(i-1)}$
DDIM Inversion $\epsilon_\theta(y^{(i-1)}, i)$
$y_1^{(i)}$
$y_3^{(i)}$
$y_3^{(i)}$
$\tau^{(T)}$
$N(0, I)$
$J(\tau^{(0)}) = \sum_k \sum_{i=0}^{T} \left\| \frac{\partial \epsilon_\theta(y_k^{(i-1)}, i)}{\partial i} \right\|_2$

Figure 4: **Compositional diffusion with Guided Search.** At each denoising timestep, CDGS iteratively denoises a batch of noisy candidate global plans by (i) **iterative resampling** to propagate information through averaged scores at overlaps (blue) and (ii) **pruning** candidates with local inconsistencies based on the predicted clean samples (yellow). This process ensures all local plans align and belong to high-likelihood regions of $p(y)$, producing globally coherent plans.

space and requires reasoning about long-horizon dependencies. Data scarcity prohibits directly learning the target global plan distribution $p(\tau)$, so a convincing alternative is to approximate it as a composition of local plan distribution $p(y)$ (using Eq. 1). Thus, one can sample short-horizon local plans $y_{1:M} \sim p(y)$ and compose them with suitable overlaps to form a coherent τ. However, as the diversity of feasible local behaviors increases, $p(y)$ becomes highly multi-modal and composing such distributions causes $p(\tau)$ to inherit combinatorial multi-modality—where each mode of the global plan distribution corresponds to a distinct sequence of modes from the local plan distribution. In this setting, naïve compositional methods ([40]) that merge distributions $y_{1:M} \sim p(y)$ via score averaging (Eq. 3) often fail due to the mode-averaging issue: selecting high-likelihood local segments that, while individually plausible, result in incompatible mode sequences—leading to inconsistent overlaps and incoherent global plans. A natural way to address multi-modality is to explore diverse modes during sampling, an idea recently explored by inference-time scaling approaches [39, 65]. However, these methods are limited to sampling from standalone distributions and not a composed sequence of distributions. The key challenge is to generate a feasible sequence of local plans that collectively form a coherent global plan—requiring a sampling algorithm that reasons over structured combinations of modes rather than collapsing into incoherent averages.

Our method: Compositional Diffusion with Guided Search (CDGS). CDGS is a structured *inference-time algorithm* designed to identify coherent sequences of local modes that form valid global plans. Specifically, CDGS employs a population-based search to explore and select promising mode sequences beyond naïve sampling. To facilitate the search, it: (i) incorporates iterative resampling into the compositional score calculation to enhance information exchange across distant segments, leading to potentially coherent global plan candidates, and (ii) prunes the incoherent candidates by evaluating the likelihood of their local segments with a ranking objective. Note that this is all within a standard denoising diffusion process, making CDGS a plug-and-play sampler applicable across domains, including robotics planning, panorama image generation, and long video generation. In the following sections, we detail each of these components and demonstrate how their integration enables efficient navigation of the complex multi-modal search space to produce coherent long-horizon plans.

3.1 Compositional Diffusion with Guided Search

A key challenge with multi-modal distributions is that naïve compositional sampling can lead to incoherent global plans: since each segment is independently sampled from $p(y)$, they may not align well at their overlaps and potentially lead to mode-averaging issues-where high-likelihood local plans do not combine to form a feasible global plan.

To address this, our approach leverages a guided search procedure that explores promising sequences of local modes while filtering out ones that are more likely to result in incoherent global plans.

Method formulation. At each diffusion timestep t, given a noisy global plan $\tau^{(t)}$, our goal is to sample from an improved next-step distribution over $\tau^{(t-1)}$, that is more likely to yield a coherent

Algorithm 1 CDGS

```
Require: Start $x_s$, Goal $x_g$, Planning horizon $H$
Require: Diffusion noise schedule,
Require: Pretrained local plan score function $\varepsilon_\theta(y^{(t)}, t)$,
Require: number of candidate plans $B$, number of elite plans $K$ at every step
 1: Initialize $B$ global plan candidates: $\tau^{(T)}$
 2: $\tau^{(T)} = (y_1^{(T)} \circ \cdots \circ y_M^{(T)}) \sim \mathcal{N}(0, \mathbf{I})$
 3: for $t = T, \ldots, 1$ do
 4:     $\varepsilon(\tau^{(t)}, t)$ = ComposedScore($\tau^{(t)}, t, \varepsilon_\theta, x_s, x_g$)
 5:     $\hat{\tau}_0^{(t)} = (\tau^{(t)} - \sqrt{1-\alpha_t}\varepsilon(\tau^{(t)}, t))/\sqrt{\alpha_t}$
 6:     Rank plans using $J(\hat{\tau}_0^{(t)})$ Eq. 5
 7:     Select best-$K$ global plans
 8:     Repopulate candidates using filtered plans
 9:     $\tau^{(t-1)} \sim p(\tau^{(t-1)} | \tau^{(t)}, \hat{\tau}_0^{(t)})$ Eq. 2
10: end for
11: return $\tau^{(0)}$
```

Algorithm 2 ComposedScore

```
Require: Noisy sample $\tau^{(t)}$, denoising timestep $t$, pretrained local plan score function $\varepsilon_\theta$
Require: Start and goal: $x_s, x_g$
Require: Number of resampling steps $U$
 1: for $u = 1, \ldots, U$ do
 2:     Calculate $\varepsilon(\tau^{(t)}, t)$ using Eq. 3
 3:     if $u < U$ then
 4:         Calculate $\tau^{(t-1)}$ using Eq. 2
 5:         Add noise to $x_s/x_g$:
 6:         $x_{s/g}^{(t-1)} \sim \mathcal{N}(\sqrt{\alpha_{t-1}} x_{s/g}, (1-\alpha_{t-1})I)$
 7:         Inpaint noisy start and goal in $\tau^{(t-1)}$
 8:         Resampling: $\tau^{(t)} \sim p(\tau^{(t)} | \tau^{(t-1)})$
 9:     end if
10: end for
11: return $\varepsilon(\tau^{(t)}, t)$
```

global plan. To achieve this, we define a modified sampling distribution:

$$p_J\left(\tau^{(t-1)} | \tau^{(t)}\right) \propto p\left(\tau^{(t-1)} | \tau^{(t)}\right) \exp\left(-J\left(\hat{\tau}_0^{(t-1)}\right)/\lambda_t\right),$$

where (i) $p(\tau^{(t-1)} | \tau^{(t)})$ is the original diffusion transition realized using the compositional score function $\varepsilon(\tau^{(t)}, t)$, (ii) $\hat{\tau}_0^{(t-1)}$ is the Tweedie-estimate of the clean global plan at timestep $t-1$, (iii) $J(\cdot)$ is a plan ranking metric we define below, and (iv) λ_t controls the exploration-exploitation tradeoff. We approximate sampling from this distribution using a Monte Carlo search procedure resembling the cross-entropy method: draw a batch of noisy global plans from $p(\tau^{(t-1)} | \tau^{(t)})$, rank them using J and retain a subset of *elite* global plans that minimizes the evaluation metric $J(\cdot)$ as illustrated in Algorithm 1. The number of elites K is a tunable parameter of our algorithm, enabling exploration of many possibilities in parallel when the planning problem is very large/difficult. Now, we just need to ensure that (i) the global plans are ranked appropriately and (ii) the candidate samples proposed by compositional sampling contain informative, globally coherent mode-sequences to pursue.

Ranking global plans via local feasibility. To guide the search effectively, we require a mechanism to evaluate the feasibility of candidate plans. Our key insight is that a global plan is feasible *iff* all of its local transitions are feasible. Since the local model $p(y)$ is trained to model feasible short-horizon behavior, high-likelihood local plans are strong indicators of local feasibility. Therefore, a globally feasible plan should consist of high-likelihood local-plan segments throughout. However, computing exact likelihoods in diffusion models is computationally expensive [51], often intractable.

To address this, we leverage DDIM inversion [50] to approximate the likelihoods of local plan segments y. Each local segment y of a sampled global plan τ goes through forward diffusion *using the learned score network* (ε_θ) such that:

$$\frac{y^{(t)}}{\sqrt{\alpha_t}} = \frac{y^{(t-1)}}{\sqrt{\alpha_{t-1}}} + \left(\sqrt{\frac{1-\alpha_t}{\alpha_t}} - \sqrt{\frac{1-\alpha_{t-1}}{\alpha_{t-1}}}\right) \varepsilon_\theta(y^{(t-1)}, t) \tag{4}$$

A high-likelihood sample follows a low-curvature path, whereas low-likelihood samples exhibit high curvature to bring noisy latents in-distribution when forward noised [17] (refer App. E). Specifically, we define a smoothness measure based on the curvature of the diffusion trajectory during inversion:

$$g\left(y^{(0)}\right) = \sum_{i=1}^{T} \left\| \frac{\partial \varepsilon_\theta(y^{(i-1)}, i)}{\partial i} \right\|_2, \quad J(\tau^{(0)}) = \prod_{m=1}^{M} \exp\left(-g\left(y_m^{(0)}\right)\right) \tag{5}$$

where $g(y^{(0)})$ measures closeness of $y^{(0)}$ to the nearest mode of $p(y)$, intuitively. A higher value of $g(y^{(0)})$ corresponds to lower-likelihood local plans. We aggregate $g(y^{(0)})$ over all local plan segments $y_{1:M}^{(0)}$ in $\tau^{(0)}$ to define the global plan ranking metric $J(\tau^{(0)})$ to measure plan feasibility. Low-quality plans have high J values, making their denoising paths more likely to be pruned.

3.2 Iterative Resampling

To ensure the effectiveness of the guided search, it is not enough to rank global plans correctly—we must also promote globally coherent candidate plans. However, standard compositional sampling fails to propagate long-horizon dependencies across overlapping local plans. Consider the running example in Fig. 3. After one denoising step, due to independent sampling of local plans, y_1 has no information about y_6, and vice versa.

To address this, we apply iterative resampling [36]: repeatedly alternating between forward noising $\tau^{(t)} \sim p(\tau^{(t)}|\tau^{(t-1)})$ and denoising steps. This procedure enables the score network's predictions for each segment to incorporate information from distant neighbors via overlapping variables, encouraging global consistency. Mathematically, this process resembles belief propagation on a chain of factors where each local plan $y_m \in y_{1:M}$ in τ depends on its neighbors y_{m-1} and y_{m+1} through the respective overlaps ($y_m \cap y_{m-1}$ and $y_m \cap y_{m+1}$). During resampling, the belief of y_m is updated as: $p(y_m|y_{m-1}, y_{m+1}) \propto p(y_m)p(y_m|y_m \cap y_{m-1})p(y_m|y_m \cap y_{m+1})$ Following Algorithm 2, after U iterations, this iterative resampling ensures that information propagates across the entire long-horizon sequence, producing a more globally coherent plan.

Summary of CDGS. We propose a guided-search algorithm by integrating a population-based pruning strategy within compositional sampling. Given a local plan score function, our approach samples potentially coherent global plan candidates and filters out plans with locally inconsistent segments. Repeating this throughout the denoising process improves the probability that the retained candidates satisfy local feasibility at every segment and are therefore globally feasible plans. Our algorithm benefits from adaptive compute at inference time, with the flexibility to scale the batch size B and the number of resampling steps U for problems with longer horizons and larger search spaces.

4 Experimental Results: Robotic Planning

In this section, we evaluate the performance of CDGS for long-horizon robotic planning. For all the experiments, we represent inputs with a low-dimensional state-space of the system comprising the pose of the end-effector and the objects in the scene in the global frame of reference. For real-world evaluations, we obtain the pose of the objects through perception, more details in App. I.

CDGS can solve learning from play problems efficiently. We evaluate CDGS for sequential-decision making tasks using the OGBench Scene task suite [44], which includes five tasks where a robot must manipulate objects (a drawer, sliding window, and cube) to reach a goal state. The primary challenge is learning from *unstructured play data* during training, which does not directly solve the target tasks. The diversity of the unstructured plans makes the local distributions highly multi-modal. We hypothesize that CDGS is an ideal method for this problem statement because it can compose short-horizon plans into meaningful long-horizon plans.

Env	Type	GCBC	GCIVL	GCIQL	HIQL	Diffuser	GSC	Ours w/o PR	Ours
OGbench scene [44]	`play`	5 ±1	42 ±4	**51** ±4	38 ±3	6 ±2	8 ±2	36 ±6	**51** ±2

Table 1: **OGbench Scene-v0: learning from play.** With much less training data requirements, CDGS performs on-par with inverse-reinforcement learning baselines and better than generative baselines like Diffuser [23] in a receding horizon control. For GSC and CDGS w/o pruning (PR), we sample the complete plan based on the oracle planning horizon.

CDGS uses a Diffuser [23] to learn the distribution of local plans (up to 4 secs of trajectory at 20 Hz) represented as a sequence of states and actions $y = \{s_1, a_1, ..., s_h, a_h\}_{i=1}^{h}$ and then composes them at inference for a given goal state to sample up to 10 secs of motion plans $\tau = \{s_i, a_i\}_{i=1}^{H}$ ($h < H$). We compare the performance CDGS with inverse reinforcement learning baselines from OGBench, including GCBC [38, 16], GCIVL, GCIQL [30], and HIQL [45], with results presented in Tab. 1.

CDGS can solve hybrid-planning problems. Task and Motion Planning (TAMP) decomposes robotic planning into a symbolic search for a sequence of discrete high-level skills (e.g., `pick`, `place`, `pull`) followed by low-level motion planning for each skill [13]. Specifically, we formulate a task-and-motion-plan as $\tau = \{y_1, \dots, y_m\}$ where $y_i = \{s_i, \pi_i, a_i, s_{i+1}\}$ where a discrete skill π_i with motion parameters a_i is executed on state s_i to get to state s_{i+1}. This entails solving a

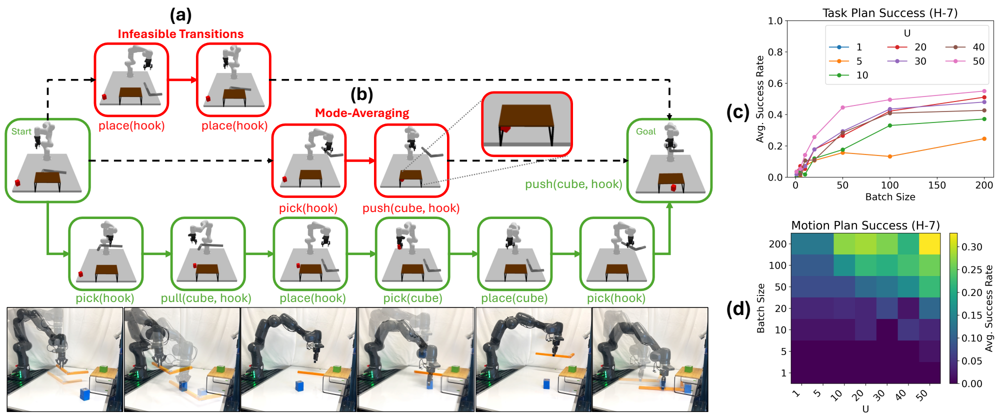


Figure 5: **Left: Visualizing plan pruning.** When compositional sampling chooses an infeasible mode sequence, the resulting plan can hallucinate out-of-distribution transitions due to **mode-averaging** as explained in Sec. 3. For instance, **(a) Infeasible transitions:** `inhand(hook)` precondition is never met for `place(hook)`, and **(b) State hallucination:** `cube` moves `under(rack)` as a result of averaging toward the goal state, despite being geometrically infeasible for `push(cube, hook)`. Our pruning objective (Eq. 5) ensures only feasible plans during denoising, where all transitions are in-distribution with our short-horizon transition diffusion model. **Right: Scaling analysis.** (H-7) denotes performance averaged over tasks of horizon 7. **(c) Task planning** success improves with batch size, with larger gains from more resampling steps. **(d) Motion planning** success improves with resampling steps, but only when batch size is large enough

hybrid-planning problem where the chosen discrete skill modes and continuous action modes must simultaneously satisfy symbolic and geometric constraints. We systematically evaluate our method on three suites of TAMP tasks, which are described in detail in App. F.

We compare our method with learning-based TAMP and other compositional methods. Specifically, we consider the following categories: (1) privileged with PDDL: **Random CEM** and **STAP CEM** [2] search for symbolic plans in a PDDL domain and apply Cross Entropy Method (CEM) optimization over potential motion plans (2) task information provided via prompting: **LLM-T2M** [33] prompts an LLM (`GPT-4.1`) and VLM (**VLM-T2M**) with descriptions of the scene along with $n = 11$ in-context examples (w/o and w/ scene images respectively) to generate a feasible task plan that is checked by a geometric motion planner (STAP CEM in this case). (3) compositional diffusion: **GSC** [40] performs compositional diffusion (equivalent to CDGS w/o RP and PR). Notably, GSC and CDGS are the *only* methods that do not rely on explicit symbolic search or LLM/VLM supervision for the task plan. The results of our evaluation are in Tab. 2.

CDGS's performance scales with compute. We hypothesize that CDGS has adaptive inference-time compute, meaning that it benefits from more compute on harder problems. We validate this hypothesis on our most challenging TAMP tasks with a planning horizon of 7. We find that increasing batch size (B) and number of resampling steps (U) increases the task planning success Fig. 5(c) and motion planning success Fig. 5(d) of CDGS. Interestingly, we find that neither increasing B nor

	Remark	**Hook Reach**		**Rearrangement Push**		**Rearrangement Memory**	
	(Task information)	**Task 1**	**Task 2**	**Task 1**	**Task 2**	**Task 1**	**Task 2**
Task Length		4	5	4	7	4	7
Random CEM	PDDL + BFS	0.14	0.10	0.08	0.00	0.02	0.02
STAP CEM		**0.66**	**0.70**	0.76	**0.70**	0.00	0.00
LLM-T2M, $n = 11$	LM Prior +	0.0	0.48	0.72	0.06	0.0	0.0
VLM-T2M, $n = 11$	Prompting	0.0	0.42	0.62	0.02	0.0	0.0
GSC	No PDDL	0.18	0.04	0.00	0.00	0.07	0.00
CDGS (w/o PR)	skill-level data only	0.24	0.12	0.12	0.00	0.11	0.00
CDGS (ours)		**0.64**	0.58	**0.84**	0.48	**0.42**	**0.18**

Table 2: **Evaluation on TAMP task-suite.** We compare CDGS with relevant search-based (PDDL Domain) and prompting based (LLM/VLM) baselines. CDGS performs on-par or slightly trails privileged methods on Hook Reach and Rearrangement Push, but substantially outperforms them on Rearrangement Memory. (success rate over 50 trials)

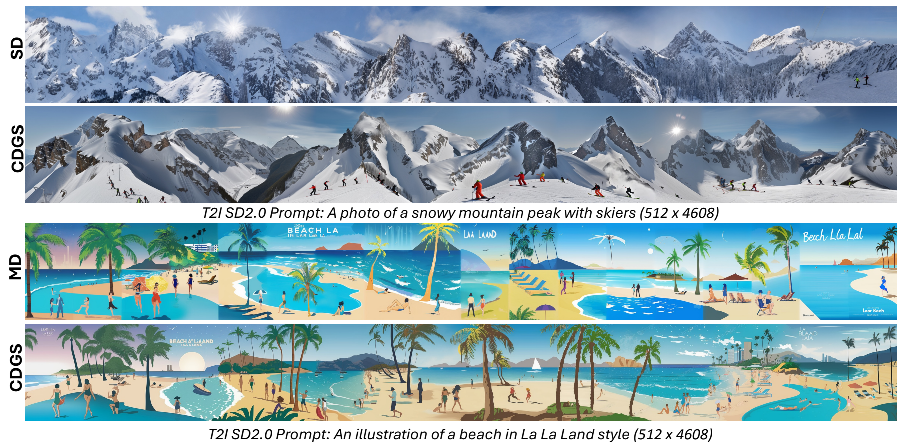

SD
CDGS
T2I SD2.0 Prompt: A photo of a snowy mountain peak with skiers (512 x 4608)
MD
CDGS
T2I SD2.0 Prompt: An illustration of a beach in La La Land style (512 x 4608)

Figure 6: **Panorama image generation.** The above figure shows the qualitative comparison of CDGS with MD [4] and SD [31]. We show qualitative intuition behind global coherence and local feasibility: while SD generates smooth panoramas, they fail to satisfy the global context (*mountain peak with skiers*), on the other hand, MD follows the global context (*beach in La La Land style*) but fails to exhibit local consistency. CDGS excels at both.

U on their own is sufficient for overall motion planning success. Thus, both resampling and pruning are essential for long-horizon tasks, as evidenced by the significant improvement of CDGS (Tab. 2).

5 CDGS FOR LONG CONTENT GENERATION

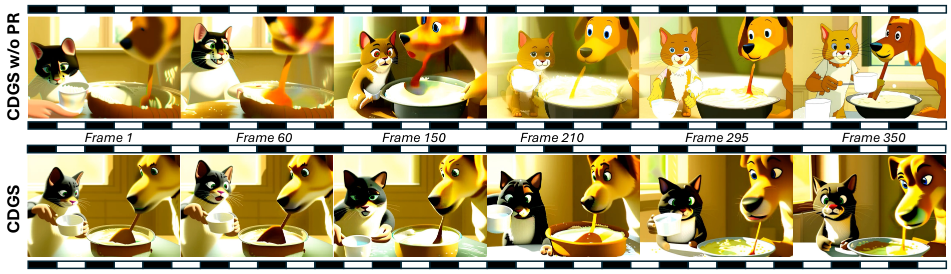
We formulate CDGS with specific design choices that enable (i) efficient message passing for global consistency and (ii) pruning denoised paths that lead to incoherent sequences. While these mechanisms are essential for long-horizon planning, we investigate their broader applicability, particularly in long-content generation tasks such as text-to-image (T2I) and text-to-video (T2V), which require spatial and temporal coherence over extended horizons. Our framework demonstrates effective improvement in long-horizon content generation.

CDGS enables coherent panoramic image generation via stitching. We evaluate CDGS on panoramic synthesis by composing multiple image patches. A panorama τ is represented as a sequence of small images y, each split into three overlapping patches $y = (x_1, x_2, x_3)$. Using Stable Diffusion-2.0 [47], we generate up to 512×4608 panoramas by stitching 512×512 images. We compare against (i) Multi-Diffusion (MD) [4], which averages scores across overlaps (image-domain analogue of GSC [40]), and (ii) Sync-Diffusion (SD) [31], which enforces LPIPS-based perceptual guidance [63]. As shown in Tab. 3, CDGS matches SD without explicit perceptual loss, indicating effective message passing for global style and perceptual transfer while maintaining prompt alignment (CLIP [46]). Qualitative samples are shown in Fig. 6, with more details in App. B.

Metric	GSC/Multi-Diffusion	Sync-Diffusion	CDGS w/o PR	CDGS
Intra-LPIPS ↓	0.72 ± 0.08	**0.58** $\pm \mathbf{0.06}$	0.61 ± 0.08	**0.59** $\pm \mathbf{0.04}$
Intra-Style-L$(\times 10^{-2})$ ↓	2.96 ± 0.24	**1.39** $\pm \mathbf{0.12}$	1.97 ± 0.08	**1.38** $\pm \mathbf{0.03}$
Mean-CLIP-S ↑	31.77 ± 2.14	31.77 ± 2.14	31.71 ± 2.34	**32.51** $\pm \mathbf{2.66}$

Table 3: **Quantitative comparison of panorama generation.** We generate 1000 panoramas of dimensions 512×4608 using 14 prompts and compare different methods based on their perceptual similarity (LPIPS [63]), style similarity (Style-loss [14]), and prompt alignment (CLIP score [46]).

CDGS can sample temporally-consistent longer videos. We follow a setup similar to panorama generation, composing shorter clips along the temporal axis for long-video generation. When short sequences of frames are stitched to make a long video, a key challenge is maintaining subject consistency and minimizing temporal artifacts. We use CogVideoX-2B [58] as the base model, capable of generating ~ 50-frame videos, and extend it to up to 350 frames at 720p resolution. We use six prompts to generate videos with naïve composition (GSC/Gen-L-Video [52] equivalent), compositional diffusion with resampling, and CDGS. The results are evaluated with VBench [21] for temporal consistency, subject fidelity, visual quality, and alignment with the prompt (refer Tab. 4).



T2V CogVideoX-2b Prompt: A cat and a dog baking a cake together in a kitchen. The cat is carefully measuring flour, while the dog is stirring the batter with a wooden spoon. The kitchen is cozy, with sunlight streaming through the window. (350 frames, 720p)

Figure 7: **Long video generation.** CDGS w/ PR (below) maintains subject-consistency while CDGS w/o PR (top) exhibits mode-averaging, resulting in significant changes to the subjects' appearances.

Qualitative analysis in Fig. 7 clearly shows the multimodal problem where multiple local plans allow satisfying the global context, but with CDGS's effect local-to-global message passing, we see an improvement in subject consistency and temporal smoothness. This comes at a minor aesthetic degradation—a tradeoff commonly observed in long-video generation models.

Method	Subject-consistency ↑	Temporal-flickering ↑	Aesthetic-quality ↑	Prompt-alignment ↑
CogVideoX-2B (50 frames)	**95.91**	**97.35**	**63.10**	**25.51**
CogVideoX-2B (350 frames)	90.24	98.44	49.44	21.78
GSC (≡ Gen-L-Video)	89.51	96.89	**60.12**	25.13
Ours w/o PR	91.06	97.08	59.40	25.42
Ours	**91.67**	**97.16**	58.90	**26.13**

Table 4: **Quantitative comparison of long-video generation.** We evaluate the performance of CDGS based on selected metrics from VBench that measure subject consistency, aesthetics, prompt alignment and temporal artifacts. We use 6 prompts (refer App. C) and generate videos with 350 frames at 720p resolution. CDGS achieves competitive video quality but for significantly (7x) extended horizons.

6 Related Work

Long-horizon content generation There are many approaches to generating long-horizon content like panoramas and long videos [25, 35, 6, 18]. Some assume access to long-horizon training data for end-to-end training [15, 5, 56, 57], while others with weaker assumptions about training data will compose the outputs of short-horizon models through outpainting [55, 27] or stitching [62, 28]. Our method belongs to the latter, enabling generalization to longer horizons than seen during training.

Generative planning. Generative models such as diffusion models[49, 20] are widely used for planning [24, 3, 7, 32, 37], though they struggle with task lengths beyond their training data. Recent works including Diffusion-CCSP [59], GSC [41], and GFC [42] have explored compositional sampling [34, 11, 62] but they sidestep the mode-averaging problem via additional mode supervision in the form of task skeletons or constraint graphs. In contrast, our approach directly addresses the mode-averaging problem to generate goal-directed long-horizon plans from short-horizon models.

Inference-time compute. Scaling inference-time computation is a powerful strategy for improving the performance of generative models [54, 43]. For diffusion models [50, 26], recent work has shown the efficacy of scaling inference-time compute through verifier-guided search during the denoising process [39, 48, 61, 64, 65]. Our algorithm differs in that it addresses the unique limitation of mode-averaging when sampling from a compositional chain of distributions.

7 Conclusion

We introduce CDGS, a framework integrating compositional diffusion with guided search to generate long-horizon sequences with short-horizon models. By embedding search within the denoising process, CDGS can handle composing highly multimodal distributions and sample solutions that are both globally coherent and locally feasible. Qualitative and quantitative results suggest that CDGS is a general pathway for extending the reach of generative models beyond their training horizons across robotic planning, panoramic images, and video generation.

8 Reproducibility statement

We are committed to ensuring that all the results presented in this paper are reproducible. To this end, we have provided pseudocodes in the paper and released the official code base through our anonymized project website: `https://cdgsearch.github.io/`. We have also provided the hyperparameters table for motion planning (refer App. G), for image generation (refer App. K) and video generation (refer App. L). Apart from this our content-generation experiments use open-source models like Stable-Diffusion-2 (refer `https://huggingface.co/stabilityai/stable-diffusion-2`) and CogVideoX-2B (refer `https://huggingface.co/zai-org/CogVideoX-2b`). For all other robotics setup, we provide more information through our project website.

References

[1] Christopher Agia, Toki Migimatsu, Jiajun Wu, and Jeannette Bohg. Taps: Task-agnostic policy sequencing. *arXiv preprint arXiv:2210.12250*, 2022.

[2] Christopher Agia, Toki Migimatsu, Jiajun Wu, and Jeannette Bohg. Stap: Sequencing task-agnostic policies. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7951–7958. IEEE, 2023.

[3] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, Tommi S. Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision making? In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=sP1fo2K9DFG`.

[4] Omer Bar-Tal, Lior Yariv, Yaron Lipman, and Tali Dekel. Multidiffusion: Fusing diffusion paths for controlled image generation. 2023.

[5] Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024.

[6] Xinyuan Chen, Yaohui Wang, Lingjun Zhang, Shaobin Zhuang, Xin Ma, Jiashuo Yu, Yali Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. Seine: Short-to-long video diffusion model for generative transition and prediction. In *The Twelfth International Conference on Learning Representations*, 2023.

[7] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[8] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

[9] Yilun Du and Leslie Pack Kaelbling. Position: Compositional generative modeling: A single model is not all you need. In *Forty-first International Conference on Machine Learning*, 2024.

[10] Yilun Du, Shuang Li, and Igor Mordatch. Compositional visual generation with energy based models. *Advances in Neural Information Processing Systems*, 33:6637–6647, 2020.

[11] Yilun Du, Conor Durkan, Robin Strudel, Joshua B Tenenbaum, Sander Dieleman, Rob Fergus, Jascha Sohl-Dickstein, Arnaud Doucet, and Will Sussman Grathwohl. Reduce, reuse, recycle: Compositional generation with energy-based diffusion models and mcmc. In *International conference on machine learning*, pp. 8489–8510. PMLR, 2023.

[12] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pp. 1515–1528. PMLR, 2018.

[13] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4(1):265–293, 2021.

[14] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2414–2423, 2016. doi: 10.1109/CVPR.2016.265.

[15] Songwei Ge, Thomas Hayes, Harry Yang, Xi Yin, Guan Pang, David Jacobs, Jia-Bin Huang, and Devi Parikh. Long video generation with time-agnostic vqgan and time-sensitive transformer. In *European Conference on Computer Vision*, pp. 102–118. Springer, 2022.

[16] Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. *arXiv preprint arXiv:1912.06088*, 2019.

[17] Alvin Heng, Harold Soh, et al. Out-of-distribution detection with a single unconditional diffusion model. *Advances in Neural Information Processing Systems*, 37:43952–43974, 2024.

[18] Roberto Henschel, Levon Khachatryan, Hayk Poghosyan, Daniil Hayrapetyan, Vahram Tadevosyan, Zhangyang Wang, Shant Navasardyan, and Humphrey Shi. Streamingt2v: Consistent, dynamic, and extendable long video generation from text. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 2568–2577, 2025.

[19] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[21] Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, et al. Vbench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21807–21818, 2024.

[22] Aapo Hyvärinen and Peter Dayan. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(4), 2005.

[23] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022.

[24] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9902–9915. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/janner22a.html`.

[25] Nikolai Kalischek, Michael Oechsle, Fabian Manhardt, Philipp Henzler, Konrad Schindler, and Federico Tombari. Cubediff: Repurposing diffusion-based image models for panorama generation. In *The Thirteenth International Conference on Learning Representations*, 2025.

[26] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in neural information processing systems*, 35: 26565–26577, 2022.

[27] Jihwan Kim, Junoh Kang, Jinyoung Choi, and Bohyung Han. Fifo-diffusion: Generating infinite videos from text without training. *Advances in Neural Information Processing Systems*, 37:89834–89868, 2024.

[28] Subin Kim, Seoung Wug Oh, Jui-Hsien Wang, Joon-Young Lee, and Jinwoo Shin. Tuning-free multi-event long video generation via synchronized coupled sampling. *arXiv preprint arXiv:2503.08605*, 2025.

[29] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[30] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[31] Yuseung Lee, Kunho Kim, Hyunjin Kim, and Minhyuk Sung. Syncdiffusion: Coherent montage via synchronized joint diffusions. *Advances in Neural Information Processing Systems*, 36:50648–50660, 2023.

[32] Wenhao Li, Xiangfeng Wang, Bo Jin, and Hongyuan Zha. Hierarchical diffusion for offline decision making. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference*

on Machine Learning, volume 202 of *Proceedings of Machine Learning Research*, pp. 20035–20064. PMLR, 23–29 Jul 2023. URL `https://proceedings.mlr.press/v202/li23ad.html`.

[33] Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. Text2motion: From natural language instructions to feasible plans. *Autonomous Robots*, 47 (8):1345–1365, 2023.

[34] Nan Liu, Shuang Li, Yilun Du, Antonio Torralba, and Joshua B Tenenbaum. Compositional visual generation with composable diffusion models. In *European Conference on Computer Vision*, pp. 423–439. Springer, 2022.

[35] Yu Lu, Yuanzhi Liang, Linchao Zhu, and Yi Yang. Freelong: Training-free long video generation with spectralblend temporal attention. *Advances in Neural Information Processing Systems*, 37:131434–131455, 2024.

[36] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11461–11471, 2022.

[37] Yunhao Luo, Utkarsh A Mishra, Yilun Du, and Danfei Xu. Generative trajectory stitching through diffusion composition. *arXiv preprint arXiv:2503.05153*, 2025.

[38] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on robot learning*, pp. 1113–1132. PMLR, 2020.

[39] Nanye Ma, Shangyuan Tong, Haolin Jia, Hexiang Hu, Yu-Chuan Su, Mingda Zhang, Xuan Yang, Yandong Li, Tommi Jaakkola, Xuhui Jia, et al. Inference-time scaling for diffusion models beyond scaling denoising steps. *arXiv preprint arXiv:2501.09732*, 2025.

[40] Utkarsh Aashu Mishra, Shangjie Xue, Yongxin Chen, and Danfei Xu. Generative skill chaining: Long-horizon skill planning with diffusion models. In *7th Annual Conference on Robot Learning*, 2023. URL `https://openreview.net/forum?id=HtJE9ly5dT`.

[41] Utkarsh Aashu Mishra, Shangjie Xue, Yongxin Chen, and Danfei Xu. Generative skill chaining: Long-horizon skill planning with diffusion models. In *Conference on Robot Learning*, pp. 2905–2925. PMLR, 2023.

[42] Utkarsh Aashu Mishra, Yongxin Chen, and Danfei Xu. Generative factor chaining: Coordinated manipulation with diffusion-based factor graph. In *8th Annual Conference on Robot Learning*, 2024. URL `https://openreview.net/forum?id=p6Wq6TjjHH`.

[43] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

[44] Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. *arXiv preprint arXiv:2410.20092*, 2024.

[45] Seohong Park, Dibya Ghosh, Benjamin Eysenbach, and Sergey Levine. Hiql: Offline goal-conditioned rl with latent states as actions. *Advances in Neural Information Processing Systems*, 36, 2024.

[46] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.

[47] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695, June 2022.

[48] Raghav Singhal, Zachary Horvitz, Ryan Teehan, Mengye Ren, Zhou Yu, Kathleen McKeown, and Rajesh Ranganath. A general framework for inference-time scaling and steering of diffusion models. *arXiv preprint arXiv:2501.06848*, 2025.

[49] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics, November 2015. URL `http://arxiv.org/abs/1503.03585`. arXiv:1503.03585 [cs].

[50] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[51] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[52] Fu-Yun Wang, Wenshuo Chen, Guanglu Song, Han-Jia Ye, Yu Liu, and Hongsheng Li. Gen-l-video: Multi-text to long video generation via temporal co-denoising. *arXiv preprint arXiv:2305.18264*, 2023.

[53] John Wang and Edwin Olson. Apriltag 2: Efficient and robust fiducial detection. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4193–4198, 2016. doi: 10.1109/IROS.2016.7759617.

[54] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[55] Chenfei Wu, Jian Liang, Xiaowei Hu, Zhe Gan, Jianfeng Wang, Lijuan Wang, Zicheng Liu, Yuejian Fang, and Nan Duan. Nuwa-infinity: Autoregressive over autoregressive generation for infinite visual synthesis. *arXiv preprint arXiv:2207.09814*, 2022.

[56] Desai Xie, Zhan Xu, Yicong Hong, Hao Tan, Difan Liu, Feng Liu, Arie Kaufman, and Yang Zhou. Progressive autoregressive video diffusion models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 6322–6332, 2025.

[57] Xin Yan, Yuxuan Cai, Qiuyue Wang, Yuan Zhou, Wenhao Huang, and Huan Yang. Long video diffusion generation with segmented cross-attention and content-rich video data curation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 3184–3194, 2025.

[58] Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. Cogvideox: Text-to-video diffusion models with an expert transformer. *arXiv preprint arXiv:2408.06072*, 2024.

[59] Zhutian Yang, Jiayuan Mao, Yilun Du, Jiajun Wu, Joshua B Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Compositional diffusion-based continuous constraint solvers. *arXiv preprint arXiv:2309.00966*, 2023.

[60] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on information theory*, 51(7):2282–2312, 2005.

[61] Jaesik Yoon, Hyeonseo Cho, Doojin Baek, Yoshua Bengio, and Sungjin Ahn. Monte carlo tree diffusion for system 2 planning. *arXiv preprint arXiv:2502.07202*, 2025.

[62] Qinsheng Zhang, Jiaming Song, Xun Huang, Yongxin Chen, and Ming-Yu Liu. Diffcollage: Parallel generation of large content with diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10188–10198, June 2023.

[63] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595, 2018.

[64] Tao Zhang, Jia-Shu Pan, Ruiqi Feng, and Tailin Wu. T-scend: Test-time scalable mcts-enhanced diffusion model. *arXiv preprint arXiv:2502.01989*, 2025.

[65] Xiangcheng Zhang, Haowei Lin, Haotian Ye, James Zou, Jianzhu Ma, Yitao Liang, and Yilun Du. Inference-time scaling of diffusion models through classical search. *arXiv preprint arXiv:2505.23614*, 2025.

[66] Yifeng Zhu, Abhishek Joshi, Peter Stone, and Yuke Zhu. Viola: Imitation learning for vision-based manipulation with object proposal priors. *arXiv preprint arXiv:2210.11339*, 2022. doi: 10.48550/arXiv.2210.11339.

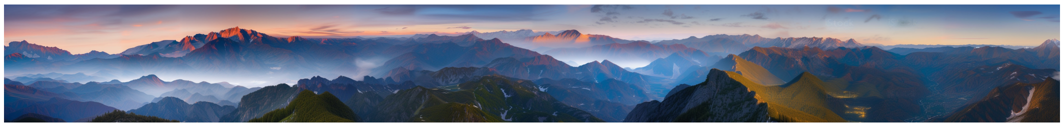
CONTENTS

A Limitations

While CDGS demonstrates strong performance in long-horizon goal-directed planning, it relies on a few simplifying assumptions that also suggest directions for future work. We assume the ability to specify a goal state, which simplifies planning but can be naturally extended to goal-generation or classifier-guided goal-conditioning methods [12]. Similarly, we generate plans for a fixed horizon, yet the framework can handle arbitrary horizons given the same start and goal, enabling selection among multiple candidate plan lengths. Finally, long-horizon dependencies are communicated through score averaging and resampling between adjacent skills; more sophisticated message-passing or attention-based mechanisms could improve efficiency and coherence across entire plans. These assumptions keep the problem tractable while providing a flexible foundation for extending CDGS to more general and complex planning scenarios.

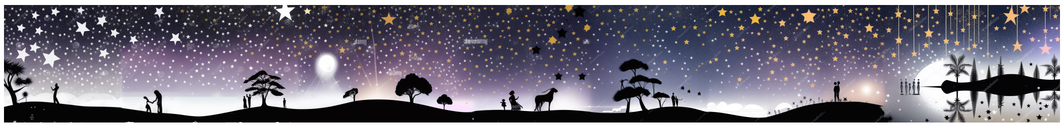
B Additional Panorama Generation Results

A photo of mountain range at twilight

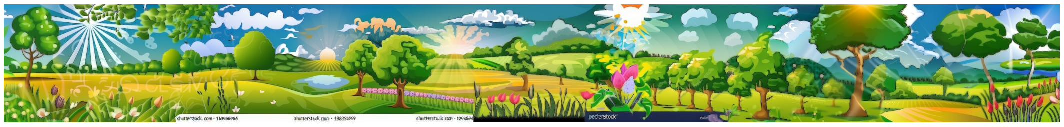


A photo of a grassland with animals

Silhouette wallpaper of a dreamy scene with shooting stars



Natural landscape in anime style illustration



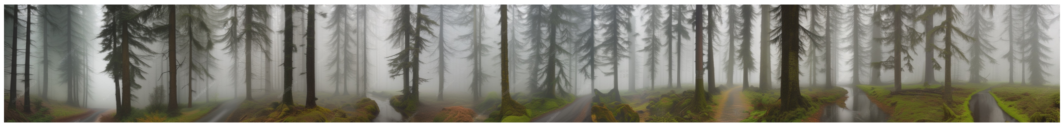
A photo of a beautiful ocean with coral reef

A photo of a lake under the northern lights

A beautiful landscape with mountains and a river

Last supper with cute corgis

A photo of a forest with a misty fog



A photo of a rock concert

C Prompts for Video Generation

We used the following standard prompts for generating the videos:

1. The camera follows behind a white vintage SUV with a black roof rack as it speeds up a steep dirt road surrounded by pine trees on a steep mountain slope, dust kicks up from it's tires, the sunlight shines on the SUV as it speeds along the dirt road, casting a warm glow over the scene. The dirt road curves gently into the distance, with no other cars or vehicles in sight. The trees on either side of the road are redwoods, with patches of greenery scattered throughout. The car is seen from the rear following the curve with ease, making it seem as if it is on a rugged drive through the rugged terrain. The dirt road itself is surrounded by steep hills and mountains, with a clear blue sky above with wispy clouds. realism, lifelike.
2. A cute happy panda, dressed in a small, red jacket and a tiny hat, sits on a wooden stool in a serene bamboo forest. The panda's fluffy paws strum a miniature acoustic guitar, producing soft, melodic tunes, move hands, singings. Nearby, a few other pandas gather, watching curiously and some clapping in rhythm. Sunlight filters through the tall bamboo, casting a gentle glow on the scene. The panda's face is expressive, showing concentration and joy as it plays. The background includes a small, flowing stream and vibrant green foliage, enhancing the peaceful and magical atmosphere of this unique musical performance. realism, lifelike.
3. A group of colorful hot air balloons take off at dawn in Cappadocia, Turkey. Dozens of balloons in various bright colors and patterns slowly rise into the pink and orange sky. Below them, the unique landscape of Cappadocia unfolds, with its distinctive 'fairy chimneys' - tall, cone-shaped rock formations scattered across the valley. The rising sun casts long shadows across the terrain, highlighting the otherworldly topography. realism, lifelike.
4. A detailed wooden toy ship with intricately carved masts and sails is seen gliding smoothly over a plush, blue carpet that mimics the waves of the sea. The ship's hull is painted a rich brown, with tiny windows. The carpet, soft and textured, provides a perfect backdrop, resembling an oceanic expanse. Surrounding the ship are various other toys and children's items, hinting at a playful environment. The scene captures the innocence and imagination of childhood, with the toy ship's journey symbolizing endless adventures in a whimsical, indoor setting. realism, lifelike.
5. A young woman with beautiful and clear eyes and blonde hair standing and white dress in a forest wearing a crown. She seems to be lost in thought, and the camera focuses on her face. The video is of high quality, and the view is very clear. High quality, masterpiece, best quality, highres, ultra-detailed, fantastic. realism, lifelike.
6. A woman walks away from a white Jeep parked on a city street at night, then ascends a staircase and knocks on a door. The woman, wearing a dark jacket and jeans, walks away from the Jeep parked on the left side of the street, her back to the camera; she walks at a steady pace, her arms swinging slightly by her sides; the street is dimly lit, with streetlights casting pools of light on the wet pavement; a man in a dark jacket and jeans walks past the Jeep in the opposite direction; the camera follows the woman from behind as she walks up a set of stairs towards a building with a green door; she reaches the top of the stairs and turns left, continuing to walk towards the building; she reaches the door and knocks on it with her right hand; the camera remains stationary, focused on the doorway; the scene is captured in real-life footage.
7. At sunset, a modified Ford F-150 Raptor roared past on the off-road track. The raised suspension allowed the huge explosion-proof tires to flip freely on the mud, and the mud splashed on the roll cage.
8. A cat and a dog baking a cake together in a kitchen. The cat is carefully measuring flour, while the dog is stirring the batter with a wooden spoon. The kitchen is cozy, with sunlight streaming through the window.

D Compositional Score Computation: CDGS's relation to existing literature

Composing the distributions defined by multiple diffusion models is well-explored in literature [11, 59]. Specifically we want to sample from the distribution of long-horizon sequences $\tau = (x_1, x_2, ..., x_N)$ by composing distributions of short-horizon sequences. There have been two main ways of composing short-horizon diffusion models in a chain:

1. Score-Averaging: approaches like GSC [41] and CDGS partition τ into overlapping segments where the score for regions of overlap can be obtained by score-averaging:

$$p(\tau) \propto \frac{p(x_1, x_2, x_3)p(x_3, x_4, x_5)\dots}{p(x_3)\dots}$$

2. Conditioning: CompDiffuser [37] partitions τ into non-overlapping segments that are conditioned on adjacent segments

$$p(\tau) \propto p(x_1|x_2)p(x_N|x_{N-1}) \prod_{i=2}^{N-1} p(x_i|x_{i-1}, x_{i+1})$$

Since CompDiffuser [37] requires training a model with conditions, we follow the more plug-n-play format of GSC [40]. For TAMP, the key difference between CDGS and GSC is that individual skill-level transitions for GSC are already conditioned on the task plan. This means that CDGS samples from the unified model $p(s_{i-1}, a_i, s_i)$ where for GSC individual segments are sampled from $p(s_{i-1}, a_i, s_i|\pi_i)$ since the oracle skill-sequence (task plan) $\pi_{1:H}$ is already provided. This greatly simplifies compositional sampling as the models in GSC only conduct motion planning, thus reducing multi-modality and mode-averaging issues significantly, whereas the models in CDGS conduct full task and motion planning.

E Pruning objective via DDIM Inversion: CDGS's relation to existing literature

DDIM Inversion is simply running the DDIM [50] denoising process backward i.e., forward noising in a deterministic way, to extract the denoising path from clean samples. Since we sample plans from a composed distribution, transition segments of a good plan should follow high-likelihood regions of the unified skill-transition distribution. A DDIM sampling based denoising looks like:

$$x^{(t-1)} = \sqrt{\alpha_{t-1}} \left(\frac{x^{(t)} - \sqrt{1-\alpha_t}\varepsilon_\theta(x_t, t)}{\sqrt{\alpha_t}} \right) + \sqrt{1-\alpha_{t-1}}\varepsilon_\theta(x^{(t)}, t)$$

We follow [17] to formulate this metric by first forward-noising each segment of the sampled plan from the task-level distribution according to:

$$\frac{x^{(t)}}{\sqrt{\alpha_t}} = \frac{x^{(t-1)}}{\sqrt{\alpha_{t-1}}} + \left(\sqrt{\frac{1-\alpha_t}{\alpha_t}} - \sqrt{\frac{1-\alpha_{t-1}}{\alpha_{t-1}}} \right) \varepsilon_\theta(x^{(t-1)}, t)$$

With $\delta_t = \sqrt{\frac{1-\alpha_t}{\alpha_t}}$ and $y^{(t)} = x^{(t)}\sqrt{1+\delta_t^2}$, we can convert the above into:

$$dy_t = \varepsilon_\theta(x^{(t-1)}, t) d\delta_t$$

Lets consider two forward-noising paths from two samples: one from high-likelihood region and one from a low-likelihood region. For both the samples, the rate of change of the integration path and its curvature directly indicate the likelihood of the clean sample. A high-likelihood sample will follow a smoother path with less curvatures while a low -likelihood sample will follow a high-curvature path to bring the noisy samples to high-likelihood regions of the noisy distribution. Hence, we consider

Taylor expansion to analyze the higher order terms:

$$\begin{aligned} y^{(t+1)} &= y^{(t)} + (\delta_{t+1}-\delta_t)\frac{dy^{(t)}}{d\delta_t}|_{(y^{(t)},t)} + (\delta_{t+1}-\delta_t)^2\frac{d^2y^{(t)}}{d\delta_t^2}|_{(y^{(t)},t)} + \dots \\ &= y^{(t)} + (\delta_{t+1}-\delta_t)\varepsilon_\theta(x^{(t-1)},t) + (\delta_{t+1}-\delta_t)^2\frac{d\varepsilon_\theta(x^{(t-1)},t)}{d\delta_t}|_{(y^{(t)},t)} + \dots \end{aligned}$$

where the second derivative term can be further decomposed into

$$\frac{d\varepsilon_\theta(x^{(t-1)},t)}{d\delta_t} = \frac{\partial\varepsilon_\theta(x^{(t-1)},t)}{\partial x^{(t-1)}}\frac{dx^{(t-1)}}{d\delta_t} + \frac{\partial\varepsilon_\theta(x^{(t-1)},t)}{\partial t}\frac{dt}{d\delta_t}$$

We find that the time-derivative term $\frac{\partial\varepsilon_\theta(x^{(t-1)},t)}{\partial t}$ is sufficient to distinguish between denoising path from high and low likelihood samples. Thus, we construct our pruning objective as:

$$g(x^{(0)}) = \sum_{t=1}^{T}\left\|\frac{\partial\varepsilon_\theta(x^{(t-1)},t)}{\partial t}\right\|_2$$

which is summing the curvature of the complete denoising timestep. A lower value of $g(x_0)$ indicates high-likelihood samples. The final objective of a sampled plan τ composing of segments $(x_1, x_2, \dots, x_H)$, where $x_k = (s_{k-1}, \pi_{k-1}, a_{k-1}, s_k)$, is calculated as:

$$\prod_{k=1}^{H}\exp\left(-g(x_k^{(0)})\right)$$

Based on the cumulative score of all segments of a plan, we select top-M plans to move on to the next denoising timestep of the compositional sampling process.

Illustrative example. We conduct a study of DDIM inversion in a 1-D setup where we consider our true distribution as the marked regions of Fig. 8(left) such that the distribution is multi-modal. We initialize a test distribution uniformly in the range $(-1,1)$. All test samples are forward integrated using the learned score function to construct the expected denoising path (as shown in Fig. 8 center red line). After we get the final latent $x^{(T)}$, we start denoising using DDIM and obtain the blue paths in Fig. 8 center. It is clearly evident that the paths from the test samples belonging to the true training distribution are the same while for OOD samples the blue lines move towards in-distribution regions.

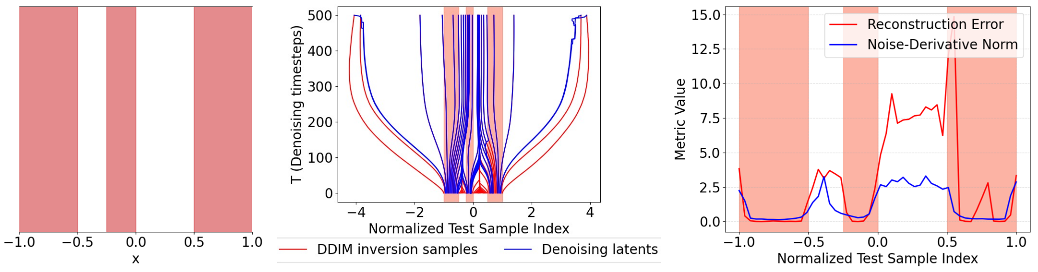


Figure 8: DDIM Inversion study in 1D distribution.

We calculate two metrics using the two calculated paths. First, we naïvely calculate the reconstruction error by calculating the cumulative difference between the latents of the red and blue path. Secondly, we calculate the metric used for CDGS's pruning objective using the derivative of the score function throughout the DDIM inversion path. Note that to calculate the later, the actual denoising path is not required. We show both metrics in Fig. 8(right) where we found that (i) although the reconstruction error differentiates the most between in- and out-of- distribution samples based on metric values, it sometimes give false positive results too (ii) the derivative based metric on the other hand is more conservative and does not require the denoising path, thus a more practical approach to take. Thus, we validate that the pruning metric is sufficient to capture samples far from the training distribution or principal modes of the learned distribution.

F ADDITIONAL TAMP SUITE DETAILS

We evaluate our framework on three task domains (`hook reach`, `rearrangement push`, and `rearrangement memory`) with two tasks each. Each of the considered suites focuses on understanding long-horizon success of one particular skill. For example, `hook reach` is about the long-term effect of executing `hook`, while `rearrangement push` focuses on `push` and `rearrangement memory` is designed to confuse the TAMP framework that perform hierarchical planning with non goal-conditioned motion planners. Each task's challenge is directly proportional to the long-horizon action dependency required to complete it. For example, `pull` affects immediately if the next skill is `pick`. But `place` affects the next skill after executing one intermediate skill (like `pick`). Similarly, action dependency is after two skills for `rearrangement push` and `rearrangement memory` tasks. We describe all of such considered tasks below.

1. **Hook Reach (Task 1):**
   - **Scene:** Table with a rack, hook, and cube
   - **Start:** Rack and hook are in workspace, cube is beyond workspace
   - **Goal:** Pick up the cube
   - **Action Skeleton:** pick(hook) $\rightarrow$ pull(cube, hook) $\rightarrow$ place(hook) $\rightarrow$ pick(cube)
2. **Hook Reach (Task 2):**
   - **Scene:** Table with a rack, hook, and cube
   - **Start:** Rack and hook are in workspace, cube is beyond workspace
   - **Goal:** Place the cube on the rack
   - **Action Skeleton:** pick(hook) $\rightarrow$ pull(cube, hook) $\rightarrow$ place(hook) $\rightarrow$ pick(cube) $\rightarrow$ place(cube, rack)

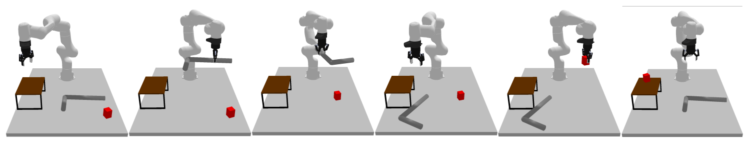


Figure 9: Hook Reach Task 2

3. **Rearrangement Push (Task 1):**
   - **Scene:** Table with a hook, cube, and rack
   - **Start:** Hook and cube are in workspace, rack is beyond workspace
   - **Goal:** Position the cube under the rack
   - **Action Skeleton:** pick(cube) $\rightarrow$ place(cube) $\rightarrow$ pick(hook) $\rightarrow$ push(cube, hook, rack)
4. **Rearrangement Push (Task 2):**
   - **Scene:** Table with a hook, cube, and rack
   - **Start:** Hook is in workspace, cube and rack are beyond workspace
   - **Goal:** Position the cube under the rack
   - **Action Skeleton:** pick(hook) $\rightarrow$ pull(cube, hook) $\rightarrow$ place(hook) $\rightarrow$ pick(cube) $\rightarrow$ place(cube) $\rightarrow$ pick(hook) $\rightarrow$ push(cube, hook, rack)

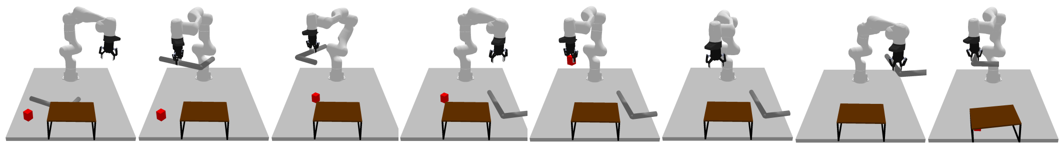


Figure 10: Rearrangement Push Task 2

5. **Rearrangement Memory (Task 1):**

- **Scene:** Table with a hook, red cube, and blue cube
- **Start:** All objects (hook, red cube, blue cube) are in workspace
- **Goal:** Put the red cube where the blue cube is
- **Action Skeleton:** pick(blue_cube) → place(blue_cube) → pick(red_cube) → place(red_cube)

6. **Rearrangement Memory (Task 2):**
   - **Scene:** Table with a hook,red cube, and blue cube
   - **Start:** Hook and blue cube are in workspace, red cube is beyond workspace
   - **Goal:** Put the red cube where the blue cube is
   - **Action Skeleton:** pick(hook) → pull(red_cube, hook) → place(hook) → pick(blue_cube) → place(blue_cube) → pick(red_cube) → place(red_cube)

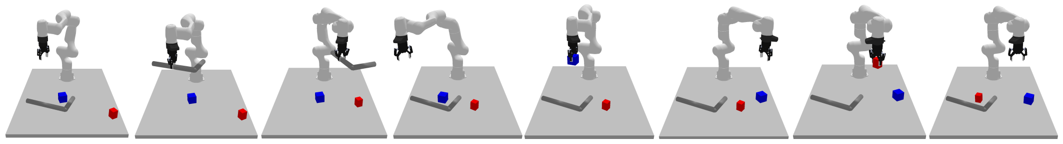


Figure 11: Rearrangement Memory Task 2

F.1 Skill Structure

We consider a finite set of parameterized skills in our skill library. The parameterization, data collection, and training method for each of the skills is described as follows:

1. `Pick`: Gripper picks up an object from the table and the parameters contain 4-DoF pose in the object's frame of reference (x, y, z, θ).
2. `Place`: Gripper places an object at the target location and parameters contain 4-DoF pose in the place target's frame of reference (x, y, z, θ). This skill requires specifying two set of parameters, the target pose and the target object (e.g. hook, table).
3. `Push`: Gripper uses the grasped object to push away another object. The skill is motivated from prior work [40, 1] where a hook object is used to `Push` blocks. The parameters of this skill are (x, y, r, θ) such that the hook is placed at the (x, y) position on the table and pushed by a distance r in the radial direction θ w.r.t. the origin of the manipulator.
4. `Pull`: Gripper uses the grasped object to pull another object inwards. The skill is also motivated from prior work [40, 1] where a hook object is used to `Pull` blocks. The parameters of this skill are (x, y, r, θ) such that the hook is placed at the (x, y) position on the table and pulled by a distance r in the radial direction θ w.r.t. the origin of the manipulator.

F.2 State Space of the Unified Skill Transition Model

CDGS assumes access to 6D object poses. In practice, we construct the system state as a concatenated vector of poses of objects present in the scenario. We use a fixed object order ([robot, rack, hook, cube1, cube2, . . .]), passing zero-vectors for absent objects, consistent across all baselines for the experiment.

G Training and Sampling: More details on Robotic Planning experiments

G.1 CDGS: Unified Score Model Training

For our TAMP suite, we collect 10000 random skill transition demonstrations for each skill by rolling out random policies in the environment. This ensures enough diversity in the system transitions in the training data. As shown in Fig. 12, we use a mixture-of-experts (MOE) model where we use N feedforward MOE layers. Each layer has a gating network and M experts, where diffusion timestep information is used through an adaptive layer normalization (AdaLN) layer. The outputs from each expert are merged using the predicted gating softmax weights to get the final score of the noisy transition tuple. For OGbnch [44], we just use the datasets provided by them: `https://github.com/seohongpark/ogbench`

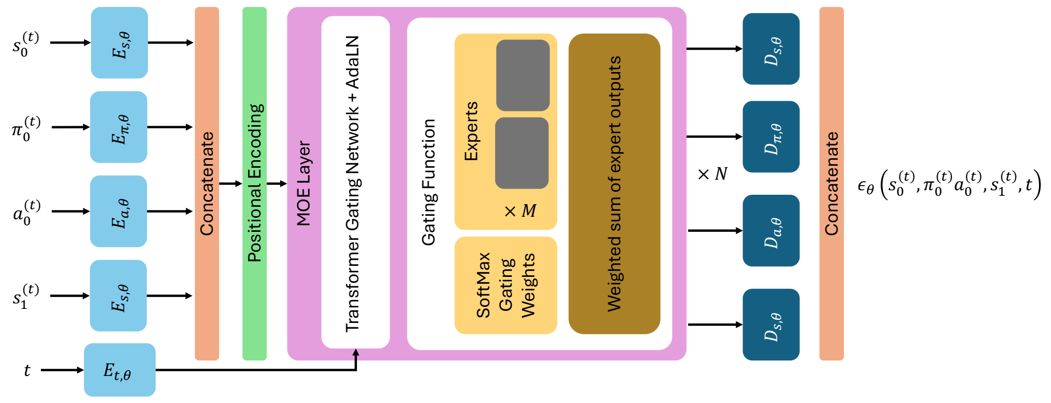


Figure 12: Network architecture for the score function

We particularly use `https://huggingface.co/docs/diffusers/en/index` library to deploy training and sampling. We provide the training hyperparameters of our setup below:

Table 5: Training setup hyperparameters for CDGS

Hyperparameter	**Value**
Num. MOE layers	3
Num. Experts per layer	6
Encoder output dim	256
Gating network Transformer num. heads	4
Hidden-dims	256
Optimizer	`torch.AdamW`
Learning rate	$1e-4$
Positional Encoding	`sinusoidal`
Num. Training Steps	$1e6$
Num Diffusion timesteps	500
Diffusion β schedule	`cosine`
Prediction type	`epsilon`

Effect of training data coverage. If we consider an "ideal" score function and a perfect representation of the system transition distributions, a solution exists if there is an overlap between the pre-condition and effect of two chosen skills that are required to solve the plan. If such an overlapping segment does not exist, CDGS will not be able to complete the plan. Hence, the training data for each skill must be diverse enough to ensure that the overlap exists. Also, it is worth noting that we use separate dataloaders for all skills to ensure equal distribution of skills in training batches and thus equal preference when sampling.

G.2 CDGS: Sampling Strategy

For the main denoising loop, we use T denoising timesteps and start with a initial batch size of B. For a plan of horizon H, we perform the compositional score computation and iterative resampling with $U(t)$ number of resampling steps at every denoising timestep t. We devise an adaptive strategy where we apply

1. no pruning for the first few denoising steps until $T_e = k_e T$. We call this as exploration phase. We keep the number of resampling iterations to low during this phase.
2. pruning starts from $T_e = k_e T$ and is done until $T_p = k_p T$. During this, at each denoising timestep, we do some resampling iterations $U(t)$ and then select top-K elites based on the pruning metric.
3. once we have potentially high-quality globally coherent sequences of local modes after a few steps of pruning, we start increasing resampling iterations. This allows us to align the local plans more closely with the optimal mode sequences.

We show the value of each hyperparameter in Tab. 6.

Table 6: TAMP suite experiments: Sampling setup hyperparameters for CDGS

Hyperparameter	**Value**
Denoising timesteps T	10
Batch size B	100 for $H = 7$ and 50 for $H = 4\&5$
Resampling schedule $U(t)$	$\frac{T-t+1}{T}(U_T)$
Maximum resampling steps U_T	50 for $H = 7$ and 40 for $H = 4\&5$
Exploration ends at k_e	0.7
Pruning ends at k_p	0.3
Top-K pruning selection	$0.2 \times B$
Pruning objective calculated with P DDIM inversion steps	$0.4 \times T$

Thus for a plan of horizon H, the total number of function evaluation (NFE) comes to be:

$$NFE = \underbrace{U_T \times \frac{T(T+1)}{2}}_{\text{Main Denoising Loop}} + \underbrace{(k_e - k_p)T \times P}_{\text{Pruning phase}}$$

Since using a single model allows batch operations of converting the B plans of horizon H into a single batched model evaluation with $B \times H$ short transitions.

G.3 CDGS: Runtime and evaluation

We observe the inference time of CDGS to be $0.5 \times H$ sec (*linear* with H) on an Nvidia L40s GPU where H is the plan length. For success metrics, we consider a task success according to the following: (1) **Hook Reach:** the cube is on rack in a stable position (2) **Rearrangement Push:** $\geq 50\%$ of the cube is under the rack and (3) **Rearrangement Memory:** cube within 0.05 m of target positions.

G.4 STAP [1]

For STAP, we use their policies, critics, and dynamics models trained with their inverse reinforcement learning pipeline (text2motion [33]) available at `https://github.com/agiachris/STAP`. For Rearrangement Push, we modify the criteria of the `Under` predicate such that $>= 50\%$ of the cube must be under the rack to be successful. We train a new model using STAP's code for `Push` and use their pre-trained models for the other skills.

G.4.1 Task Planning

STAP by itself is only a motion planner. In order to solve full TAMP problems, it must be integrated with an external task planner. Symbolically-feasible skill sequences found by the task planner are evaluated and ranked by STAP for geometric feasibility. For our experiments, we use a BFS-based symbolic planner that searches through a hand-designed PDDL domain.

To better reflect practical considerations, we design the PDDL domain for each task so that provided geometric information is minimized while ensuring that the correct task plan can always be found. We modify the BFS algorithm so that it can revisit previously visited states as the hidden geometric predicates may be different despite the same symbolic predicates.

Remark on Rearrangement Memory task. There are two particular characteristics required in a TAMP to solve Rearrangement Memory task:

1. The symbolic planner must understand which particular symbolic state will satisfy the goal condition. Since most required skills are `pick` and `place`, the symbolic effect of all `place` actions are same. As the exact goal position is not embed in the symbolic states, it is not possible for a naïve task planner to solve for a skill sequence.
2. The task planner can give many feasible solutions that the motion planner must evaluate to find the final task and motion plan. This requires goal-conditioned planners. Since STAP uses Q-function based value estimates to evaluate plans, we find that it struggles with the task as it is not a goal-conditioned method.

G.4.2 CEM Sampling

The STAP baselines use a CEM-based sampling algorithm. An initial prior for the actions is sampled for the start state, and then optimized using the value and dynamics models with CEM-optimization. The only difference between Random CEM and STAP CEM is that Random CEM samples the prior from a uniform distribution? (double-check) while STAP samples the prior from its learned policy models

To make a fair comparison between CDGS's diffusion-based sampling and STAP's CEM-based sampling, we match the sampling budget based on the number of function evaluations. Since our unified model serves the same purpose as STAP's policy, value, and dynamics models, we consider evaluating one set of STAP's policy, value, and dynamics models for a skill to be one function evaluation. For STAP, the CEM runs num iterations of sampling for $N * batch_sizesamples$. Thus, we match the number of sampling iterations and batch size. The exact budgets for each task are given below.

Table 7: CEM-sampling parameters for STAP

Task	Samples	Iterations	Elites	Total NFE	CDGS NFE
Hook Reach 1	40	132	16	5280	2300
Hook Reach 2	50	165	20	8250	2300
Rearrangement Push 1	50	165	20	8250	2300
Rearrangement Push 2	70	336	28	23520	2850
Rearrangement Memory 1	40	132	16	5280	2300
Rearrangement Memory 2	70	336	28	23520	2850

G.4.3 Uncertainty Quantification

The LLM Planner in text2motion [33] can sometimes generate symbolically invalid actions i.e. (place(cube) when nothing is in hand), which are out-of-distribution for the learned models. text2motion uses a simple ensemble-based OOD detection method (detailed in appendix A.2 of their paper) to filter out symbolically-invalid actions. We use this for all text2motion baselines. For completeness, we also include this in STAP's baselines as **STAP CEM + UQ**, but it does not make any significant improvements.

H ADDITIONAL ABLATIONS

H.1 CDGS: A BETTER PRIOR FOR TASK-LEVEL TRAJECTORY SAMPLING

The proposed method constructs a task-level distribution from skill-level distribution given the current state, the intended goal state and the planning horizon. Specifically, CDGS finds a sequence of modes with overlapping pre-condition and effects by systematic exploration and pruning. While this does not always ensure that the plan is symbolically-geometrically feasible, we observe that choosing the top two plans and expanding the BFS tree with system rollouts leads to higher success rates. As shown in Tab. 8, the CDGS (BFS-2) proves to be an upper bound of our approach. This points out that CDGS constructs meaningful task-level distribution with correct task plans.

Table 8: The success rate of the proposed CDGS algorithm is shown and compared with a variant that performs BFS with the top-2 skill chains at every step and uses system dynamics to rollout. All results are calculated from 50 trials for each task.

	Hook Reach		Rearrangement Push		Rearrangement Memory	
	Task 1	Task 2	Task 1	Task 2	Task 1	Task 2
Task Length	4	5	4	7	4	7
Full Generative TAMP (no PDDL, skill-level data only)						
CDGS (ours)	0.64	0.58	0.84	0.48	0.42	0.18
Full Generative TAMP (no PDDL, skill-level data only) + Rollout with system dynamics						
CDGS (BFS-2)	0.72	0.64	0.90	0.62	0.48	0.22

H.2 ANALYZING SCALING FOR INDIVIDUAL TASKS

We analyze how varying the batch size B and the number of resampling iterations U affects overall planning performance across all long-horizon tasks of horizon (H) 4&5 (Hook Reach Task 1

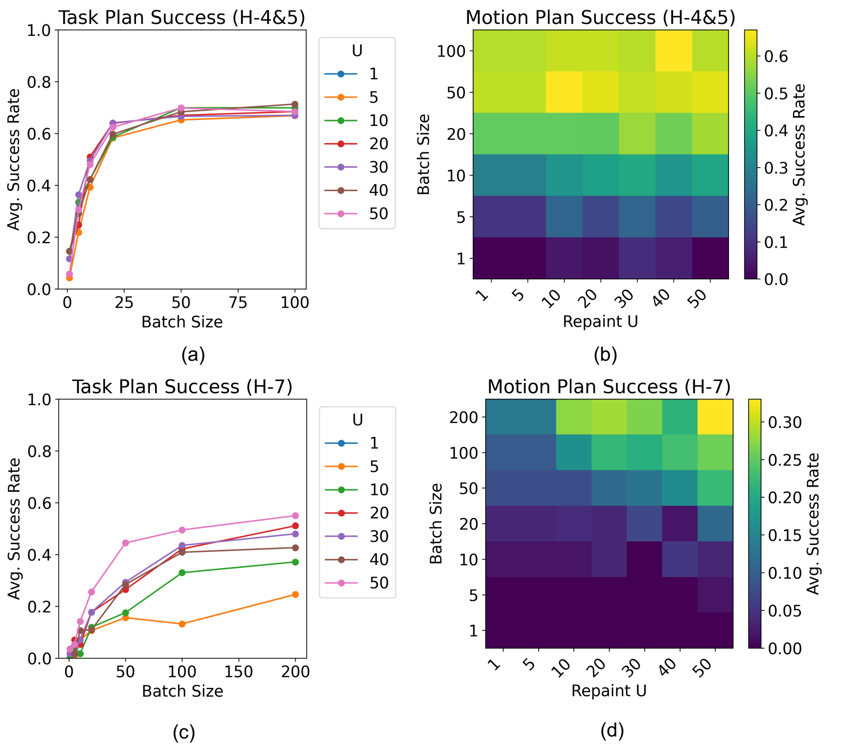


Figure 13: We show the effect of scaling B and U on the overall task planning and motion planning success of CDGS for shorter tasks ($H = 4\&5$) in (a,b) and longer tasks ($H = 7$) in (c,d)

and Task 2; Rearrangement Push Task 1; Rearrangement Memory Task 1) and longer $H = 7$ (Rearrangement Push Task 2; Rearrangement Memory Task 2) in Fig. 13. As shown in Fig. 13(a, c), increasing B yields a clear, monotonic rise in task-planning success: larger candidate sets diversify the search over the task-level distribution and enable the pruning stage to more reliably identify viable skill sequences. Motion-planning success exhibits a similar trend in Fig. 13(b, d), demonstrating that a more diverse initial sample pool benefits the motion-planning optimization as well. We can also seeFig. 13(b, d) that at lower batch size, increasing the number of resampling steps yields only marginal improvements: without pruning, repeated denoising can still suffer from mode-averaging local minima, where incorrect skill sequences become self-reinforcing. It is only when resampling is coupled with pruning that results in better task planning as well as permit bidirectional "message-passing" of information between the start and goal states—compensating for temporal misalignments at skill (pre-condition and effect) intersection—and thereby unlock significant gains in both task and motion success rates.

I Hardware Setup

The experimental setup, illustrated in Fig. 14, consists of the same Franka Panda robot arm, several blocks, a rack, and a hook, observed by an Azure Kinect camera. The camera is mounted in an inclined front-view configuration. AprilTag [53] (`https://github.com/fabrizioschiano/apriltag2`) markers are used for SE(3) pose detection. We employ Deoxys [66] (`https://github.com/UT-Austin-RPL/deoxys_control`) for control. After obtaining the SE(3) poses of all the objects: (1) we construct the same environment in simulation, (2) deploy our algorithm in simulation, and (3) execute the planned action in real environment and finally replan.

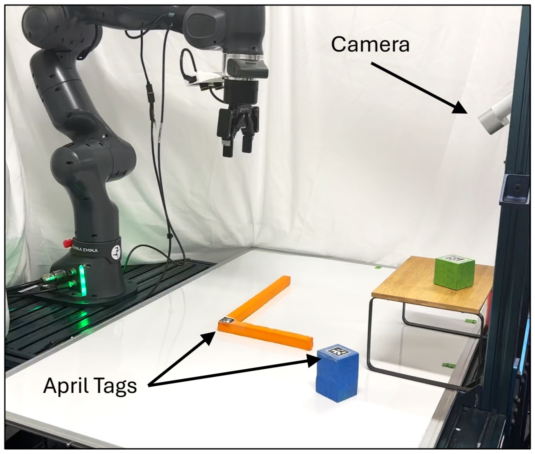


Figure 14: Hardware setup

J LLM AND VLM PROMPTING

J.1 LLM PROMPTING

To make the fairest comparison, we use the same prompt style and in-context examples as text2motion[33], which can be found in Appendix B.2. of their paper. We find that having the model generate multiple candidate task-plans during the shooting phase is critical to task performance, so we add a minimal system prompt to make the LLM instruction-following explicit. An example of a full prompt for **LLM-T2M,** $n = 1$ for `hook Reach` Task 1 is shown below.

User Prompt

Respond directly in the format specified in the output format section, following the instructions exactly for how many sequences to generate i.e. generate 5 sequences if asked for the ”Top 5 robot action sequences”.

Available primitives: [’pick(a)’, ’place(a, b)’, ’pull(a, hook)’, ’push(a, hook, rack)’]
Available predicates: [’on(a, b)’, ’inhand(a)’, ’under(a, b)’]
Available scene objects: [’table’, ’blue_box’, ’cyan_box’, ’hook’, ’rack’, ’red_box’, ’yellow_box’]
Object relationships: [’inhand(hook)’, ’on(red_box, rack)’, ’on(yellow_box, table)’, ’on(blue_box, table)’, ’on(cyan_box, rack)’, ’on(rack, table)’]
Human instruction: could you move all the boxes onto the rack?
Goal predicate set: [[’on(yellow_box, rack)’, ’on(blue_box, rack)’]]
Top 1 robot action sequences: [’pull(yellow_box, hook)’, ’place(hook, table)’, ’pick(yellow_box)’, ’place(yellow_box, rack)’, ’pick(blue_box)’, ’place(blue_box, rack)’]

Available scene objects: [’table’, ’rack’, ’hook’, ’red_box’]
Object relationships: [’on(rack, table)’, ’on(hook, table)’, ’on(red_box, table)’]
Human instruction: How would you get the red box in the robot’s hand?
Goal predicate set: [[’inhand(red_box)’]]
Top 5 robot action sequences (python list of lists):

Model Completion

[[’pick(red_box)’], [’push(red_box, hook, rack)’, ’pick(red_box)’], [’pick(hook)’, ’place(hook, table)’, ’pick(red_box)’], [’pick(hook)’, ’place(hook, rack)’, ’pick(red_box)’], [’pick(hook)’, ’place(hook, table)’, ’push(red_box, hook, rack)’, ’pick(red_box)’]]

Interestingly, even though many of the prompts contain partial-to-complete solutions in the in-context examples like the one above, we find that this does not guarantee the LLM will generate the correct plan.

A key piece to text2motion’s success for planning despite LLM’s lack of geometric awareness is the alternation between shooting and greedy search. When the shooting motion planner fails to find a geometrically feasible motion plan from the 5 task plan candidates, the algorithm falls back to a say-can style greedy search that executes the next action deemed most feasible by a combination of the LLM task planner and the motion planner. This allows the LLM task planner to explore with geometric feedback, and is the reason why text2motion is able to solve `Rearrangement Push` Task 1. For our evaluations, we allow 10 tries, which is much more than the steps required to solve any of the tasks in our evaluation suites.

J.2 VLM Prompting

For VLM experiments, we modify the system prompt and insert scene images before the scene description. Below is an example of `Hook Reach` Task 1 with $n = 11$ in-context examples.

User Prompt

Respond directly in the format specified in the output format section, following the instructions exactly for how many sequences to generate i.e. generate 5 sequences if asked for the ”Top 5 robot action sequences”. Review the provided images carefully when constructing your plan.
Available primitives: [’pick(a)’, ’place(a, b)’, ’pull(a, hook)’, ’push(a, hook, rack)’]
Available predicates: [’on(a, b)’, ’inhand(a)’, ’under(a, b)’]
Available scene objects: [’table’, ’hook’, ’rack’, ’yellow_box’, ’blue_box’, ’red_box’]
Object relationships: [’inhand(hook)’, ’on(yellow_box, table)’, ’on(rack, table)’, ’on(blue_box, table)’]
Human instruction: How would you push two of the boxes to be under the rack?
Goal predicate set: [[’under(yellow_box, rack)’, ’under(blue_box, rack)’], [’under(blue_box, rack)’, ’under(red_box, rack)’], [’under(yellow_box, rack)’, ’under(red_box, rack)’]]
Top 1 robot action sequences: [’push(yellow_box, hook, rack)’, ’push(red_box, hook, rack)’]

Available scene objects: [’table’, ’blue_box’, ’cyan_box’, ’hook’, ’rack’, ’red_box’, ’yellow_box’]
Object relationships: [’inhand(hook)’, ’on(red_box, rack)’, ’on(yellow_box, table)’, ’on(blue_box, table)’, ’on(cyan_box, rack)’, ’on(rack, table)’]
Human instruction: could you move all the boxes onto the rack?
Goal predicate set: [[’on(yellow_box, rack)’, ’on(blue_box, rack)’]]
Top 1 robot action sequences: [’pull(yellow_box, hook)’, ’place(hook, table)’, ’pick(yellow_box)’, ’place(yellow_box, rack)’, ’pick(blue_box)’, ’place(blue_box, rack)’]

Available scene objects: [’table’, ’blue_box’, ’hook’, ’rack’, ’red_box’, ’yellow_box’]
Object relationships: [’on(hook, table)’, ’on(red_box, table)’, ’on(blue_box, table)’, ’on(yellow_box, rack)’, ’on(rack, table)’]
Human instruction: Move the ocean colored box to be under the rack and ensure the hook ends up on the table.
Goal predicate set: [[’under(blue_box, rack)’]]
Top 1 robot action sequences: [’pick(red_box)’, ’place(red_box, table)’, ’pick(yellow_box)’, ’place(yellow_box, rack)’, ’pick(hook)’, ’push(blue_box, hook, rack)’, ’place(hook, table)’]

Available scene objects: [’table’, ’cyan_box’, ’hook’, ’red_box’, ’yellow_box’, ’rack’, ’blue_box’]
Object relationships: [’on(hook, table)’, ’on(red_box, table)’, ’on(blue_box, table)’, ’on(cyan_box, table)’, ’on(rack, table)’, ’under(yellow_box, rack)’]
Human instruction: How would you get the cyan box under the rack and then ensure the hook is on the table?
Goal predicate set: [[’under(cyan_box, rack)’, ’on(hook, table)’]]
Top 1 robot action sequences: [’pick(blue_box)’, ’place(blue_box, table)’, ’pick(red_box)’, ’place(red_box, table)’, ’pick(hook)’, ’push(cyan_box, hook, rack)’, ’place(hook, table)’]

Interestingly, we find that including images in the prompt degrades the performance.

User Prompt

Available scene objects: ['table', 'cyan_box', 'hook', 'blue_box', 'rack', 'red_box']
Object relationships: ['on(hook, table)', 'on(rack, table)', 'on(blue_box, table)', 'on(cyan_box, table)', 'on(red_box, table)']
Human instruction: How would you push all the boxes under the rack? Goal predicate set: [['under(blue_box, rack)', 'under(cyan_box, rack)', 'under(red_box, rack)']]
Top 1 robot action sequences: ['pick(blue_box)', 'place(blue_box, table)', 'pick(hook)', 'push(cyan_box, hook, rack)', 'place(hook, table)', 'pick(blue_box)', 'place(blue_box, table)', 'pick(hook)', 'push(blue_box, hook, rack)', 'push(red_box, hook, rack)']

Available scene objects: ['table', 'cyan_box', 'hook', 'rack', 'red_box', 'blue_box']
Object relationships: ['on(hook, table)', 'on(cyan_box, rack)', 'on(rack, table)', 'on(red_box, table)', 'inhand(blue_box)']
Human instruction: How would you set the red box to be the only box on the rack?
Goal predicate set: [['on(red_box, rack)', 'on(blue_box, table)', 'on(cyan_box, table)']]
Top 1 robot action sequences: ['place(blue_box, table)', 'pick(hook)', 'pull(red_box, hook)', 'place(hook, table)', 'pick(red_box)', 'place(red_box, rack)', 'pick(cyan_box)', 'place(cyan_box, table)']

Available scene objects: ['table', 'cyan_box', 'red_box', 'hook', 'rack']
Object relationships: ['on(hook, table)', 'on(rack, table)', 'on(cyan_box, rack)', 'on(red_box, rack)']
Human instruction: put the hook on the rack and stack the cyan box above the rack - thanks
Goal predicate set: [['on(hook, rack)', 'on(cyan_box, rack)']]
Top 1 robot action sequences: ['pick(hook)', 'pull(cyan_box, hook)', 'place(hook, rack)', 'pick(cyan_box)', 'place(cyan_box, rack)']

Available scene objects: ['table', 'cyan_box', 'hook', 'rack', 'red_box', 'blue_box']
Object relationships: ['on(hook, table)', 'on(blue_box, rack)', 'on(cyan_box, table)', 'on(red_box, table)', 'on(rack, table)']
Human instruction: Move the warm colored box to be underneath the rack.
Goal predicate set: [['under(red_box, rack)']]
Top 1 robot action sequences: ['pick(blue_box)', 'place(blue_box, table)', 'pick(red_box)', 'place(red_box, table)', 'pick(hook)', 'push(red_box, hook, rack)']

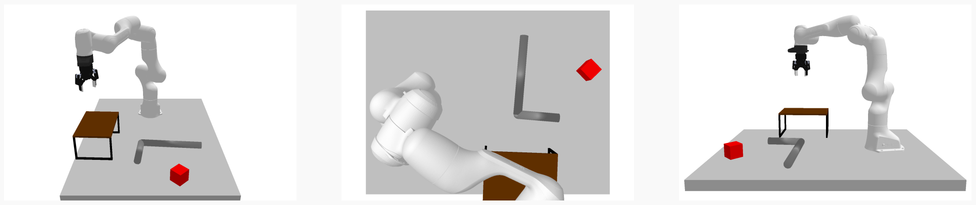
Available scene objects: ['table', 'blue_box', 'red_box', 'hook', 'rack', 'yellow_box']
Object relationships: ['on(hook, table)', 'on(blue_box, table)', 'on(rack, table)', 'on(red_box, table)', 'on(yellow_box, table)']
Human instruction: situate an odd number greater than 1 of the boxes above the rack
Goal predicate set: [['on(blue_box, rack)', 'on(red_box, rack)', 'on(yellow_box, rack)']]
Top 1 robot action sequences: ['pick(hook)', 'pull(blue_box, hook)', 'place(hook, table)', 'pick(blue_box)', 'place(blue_box, rack)', 'pick(red_box)', 'place(red_box, rack)', 'pick(yellow_box)', 'place(yellow_box, rack)']

User Prompt

continued...

Available scene objects: ['table', 'cyan_box', 'hook', 'yellow_box', 'blue_box', 'rack']
Object relationships: ['on(hook, table)', 'on(yellow_box, rack)', 'on(rack, table)', 'on(cyan_box, rack)']
Human instruction: set the hook on the rack and stack the yellow box onto the table and set the cyan box on the rack
Goal predicate set: [['on(hook, rack)', 'on(yellow_box, table)', 'on(cyan_box, rack)']]
Top 1 robot action sequences: ['pick(yellow_box)', 'place(yellow_box, table)', 'pick(hook)', 'pull(yellow_box, hook)', 'place(hook, table)']

Available scene objects: ['table', 'rack', 'hook', 'cyan_box', 'yellow_box', 'red_box']
Object relationships: ['on(yellow_box, table)', 'on(rack, table)', 'on(cyan_box, table)', 'on(hook, table)', 'on(red_box, rack)']
Human instruction: Pick up any box.
Goal predicate set: [['inhand(yellow_box)'], ['inhand(cyan_box)']] Top 1 robot action sequences: ['pick(yellow_box)']



Available scene objects: ['table', 'rack', 'hook', 'red_box']
Object relationships: ['on(rack, table)', 'on(hook, table)', 'on(red_box, table)']
Human instruction: How would you get the red box in the robot's hand? Goal predicate set: [['inhand(red_box)']]
Top 5 robot action sequences (python list of lists):

Model Completion

[['pick(red_box)'], ['pick(hook)', 'place(hook, table)', 'pick(red_box)'], ['pick(rack)', 'place(rack, table)', 'pick(red_box)'], ['pick(hook)', 'place(hook, table)', 'pick(rack)', 'place(rack, table)', 'pick(red_box)'], ['pick(red_box)', 'place(red_box, table)', 'pick(red_box)']]

K Pseudo-code and Hyperparameters for Image Generation

```
@torch.no_grad()
def text2panorama_noise_resample_pruning(self, prompts,
    height=512, width=2048, num_inference_steps=50,
    guidance_scale=7.5, num_samples_per_prompt=10, top_K=0.2
):

    # Prompts -> text embeds
    text_embeds = self.get_text_embeds(prompts)

    # Define panorama grid and get views for individual segments
    views, covered_width = get_views_gtamp(height, width)
    latent = torch.randn((
        num_samples_per_prompt,
        self.unet.in_channels,
        height // 8,
        covered_width
    ), device=self.device)
    count = torch.zeros_like(latent)
    value = torch.zeros_like(latent)

    self.scheduler.set_timesteps(num_inference_steps)

    with torch.autocast('cuda'):
        num_timesteps = len(self.scheduler.timesteps)
        for i, t in enumerate(tqdm(self.scheduler.timesteps)):
            U = int(
                min(
                    max(
                        (float(i) / float(len(self.scheduler.timesteps))) * \
                        self.num_resampling_steps,
                        5
                    ),
                self.num_resampling_steps
                )
            )
            for u in tqdm(range(U), leave=False):
                count.zero_()
                value.zero_()

                all_latents = []

                for h_start, h_end, w_start, w_end in views:
                    latent_view = latent[:, :, h_start:h_end, w_start:w_end]
                    all_latents.append(latent_view)

                latent_view = torch.stack(all_latents, dim=0) # [N, B, C, H, W]
                N, B = latent_view.shape[0], latent_view.shape[1]
                latent_view_batched = latent_view.view(
                    -1,
                    *latent_view.shape[2:]
                ) # [N*B, C, H, W]

                positive_text_embeds, negative_text_embeds = text_embeds.chunk(2)
                positive_text_embeds = positive_text_embeds.repeat(N*B, 1, 1)
                negative_text_embeds = negative_text_embeds.repeat(N*B, 1, 1)
                text_embeds_batched = torch.cat([
                    positive_text_embeds,
                    negative_text_embeds
                ], dim=0) # [2*N*B, 77, 768]

                latent_model_input = torch.cat([latent_view_batched] * 2, dim=0)

                noise_pred = self.unet(
                    latent_model_input,
                    t,
                    encoder_hidden_states=text_embeds_batched)['sample']

                # perform guidance
                noise_pred_uncond, noise_pred_cond = noise_pred.chunk(2)
                noise_pred = noise_pred_uncond + \
                    guidance_scale * (noise_pred_cond - noise_pred_uncond)

                noise_pred_batched = noise_pred.view(
                    N, B,
                    noise_pred.shape[-3],
                    noise_pred.shape[-2], noise_pred.shape[-1]
                ) # [N, B, C, H, W]

                for idx, (h_start, h_end, w_start, w_end) in enumerate(views):
                    noise_pred_batched_view = noise_pred_batched[idx] # [B, C,
                    # compute the denoising step with the reference model
                    value[:, :, h_start:h_end, w_start:w_end] += noise_pred_batched_view
                    count[:, :, h_start:h_end, w_start:w_end] += 1

                noise_combined = torch.where(count > 0, value / count, value)
                latent = self.scheduler.step(noise_combined, t, latent)

                if u < U-1 and i < len(self.scheduler.timesteps)-1 and i > 0:
                    pred_x0 = latent['pred_original_sample']
                    latent = latent['prev_sample']
                    latent = self.undo_step(latent, pred_x0, noise_combined, t)
                elif u == U-1 and \
                    (i < 0.4*num_timesteps and i > 0.1*num_timesteps):
                    pred_x0 = latent['pred_original_sample']
                    latent = latent['prev_sample']
                    latent = self.inversion_pruning(
                        pred_x0,
                        latent,
                        text_embeds_batched,
                        views,
                        guidance_scale,
                        top_K
                    )
                else:
                    latent = latent['prev_sample']

    return latent
```

```python
def inversion_pruning(self, pred_x0, latents, text_embeds, views,
    guidance_scale, top_K
):

    num_models = len(views)
    B = pred_x0.shape[0]
    all_timesteps = self.scheduler.timesteps.flip(dims=(0,))
    num_inference_steps = len(all_timesteps)

    batched_x0s = []
    for h_start, h_end, w_start, w_end in views:
        batched_x0s.append(pred_x0[:, :, h_start:h_end, w_start:w_end])

    batched_x0s = torch.stack(batched_x0s, dim=0)  # [num_models, N, C, H, W]
    batched_x0s = batched_x0s.view(
        num_models * B, -1,
        batched_x0s.shape[-2],
        batched_x0s.shape[-1]
    )

    inversion_latents = batched_x0s.clone()
    all_noise_prediction = []

    for idx, i in tqdm(
        enumerate(all_timesteps[:-num_inference_steps//2+1]),
        leave=False,
        total=num_inference_steps-1
    ):
        t = i
        t_next = all_timesteps[idx + 1]
        alpha_t = self.scheduler.alphas_cumprod[t]
        alpha_t_next = self.scheduler.alphas_cumprod[t_next]
        sqrt_alpha_t = torch.sqrt(alpha_t)
        sqrt_alpha_t_next = torch.sqrt(alpha_t_next)
        sqrt_one_minus_alpha_t = torch.sqrt(1 - alpha_t)
        sqrt_one_minus_alpha_t_next = torch.sqrt(1 - alpha_t_next)

        with torch.no_grad():
            latent_model_input = torch.cat([inversion_latents] * 2)
            noise_pred = self.unet(
                latent_model_input,
                t,
                encoder_hidden_states=text_embeds
            )['sample']
            noise_pred_uncond, noise_pred_cond = noise_pred.chunk(2)
            noise_pred_combined = noise_pred_uncond + \
                guidance_scale * (noise_pred_cond - noise_pred_uncond)

            x0_pred = (inversion_latents - \
                sqrt_one_minus_alpha_t * noise_pred_combined) / sqrt_alpha_t
            x0_pred = torch.clamp(x0_pred, -1.0, 1.0)
            noise_pred_combined = (inversion_latents - \
                sqrt_alpha_t * x0_pred) \
                / sqrt_one_minus_alpha_t
            inversion_latents = sqrt_alpha_t_next * x0_pred + \
                sqrt_one_minus_alpha_t_next * noise_pred_combined
            all_noise_prediction.append(noise_pred_combined)

    all_intermediate_noise_preds = torch.stack(all_noise_prediction, dim=1)
    derivative = torch.diff(all_intermediate_noise_preds, dim=1)

    all_scores = torch.norm(
        derivative.reshape(num_models*B, -1),
        dim=1
    ).reshape(num_models, B)
    final_scores = all_scores.mean(dim=0) # (B,)

    num_selected_samples = max(int(top_K * B), 1)
    topk_indices = torch.topk(
        final_scores,
        k = num_selected_samples,
        largest=False
    )[1]

    arranged_batch = latents.clone()
    arranged_batch = arranged_batch[topk_indices]

    while arranged_batch.shape[0] < B:
        arranged_batch = torch.cat([arranged_batch, arranged_batch], dim=0)

    arranged_batch = arranged_batch[:B]

    return arranged_batch
```

Table 9: Sampling setup hyperparameters for panorama generation experiments

Hyperparameter	**Value**
Denoising timesteps T	50
Batch size B	10
Composition weights $\gamma_{1:H}$	0.5
Resampling schedule $U(t)$	$\frac{T-t+1}{T}(U_T)$
Maximum resampling steps U_T	10
Exploration ends at k_e	0.2
Pruning ends at k_p	0.5
Top-K pruning selection	$0.4 \times B$
Pruning objective calculated with P DDIM inversion steps	$0.5 \times T$

All experiments were run on single NVIDIA™ L40s or NVIDIA™ A100 GPUs.

L Pseudo-code and Hyperparameters for Video Generation

We modify CogVideoX pipeline provided in Huggingface: `https://github.com/huggingface/diffusers/blob/v0.35.1/src/diffusers/pipelines/cogvideo/pipeline_cogvideox.py`.

We keep the logic same as images.

Table 10: Sampling setup hyperparameters for long-video generation experiments

Hyperparameter	**Value**
Denoising timesteps T	30
Batch size B	10
Composition weights $\gamma_{1:H}$	0.5
Resampling schedule $U(t)$	$\frac{T-t+1}{T}(U_T)$
Maximum resampling steps U_T	10
Exploration ends at k_e	0.3
Pruning ends at k_p	0.6
Top-K pruning selection	$0.4 \times B$
Pruning objective calculated with P DDIM inversion steps	$0.5 \times T$

All experiments were run on single NVIDIA™ H100 GPUs.