



Figure 3: (a) Maze environment. (b) Robotic manipulation task. (c) Visual maze environment.

## A Experiment details

### A.1 Computation resources

All experiments were conducted on high-performance hardware consisting of 8 NVIDIA RTX 4090 GPUs, 512GB system memory, and a 96-thread CPU. Training each model required about 6 hours, while inference for comprehensive experimental evaluation took up to 1 hour in the most computationally intensive cases.

### A.2 Environment details

To evaluate our C-MCTD, we utilized the same benchmarks as described in Yoon et al. [30]. For robot arm manipulation and visual maze tasks, we maintained identical environment settings as the previous work and therefore omitted detailed discussion of these environments. For the long-horizon maze environments, we specifically employed the *stitch* datasets rather than the navigate datasets [26]. This choice creates a more challenging evaluation scenario that requires generating plans significantly longer than the trajectories observed during training, for instance, while the trajectory length in the training dataset is limited to 200 steps, successfully reaching goals in test environments requires generating plans up to 1000 steps in length. This setting allows us to assess the extendable planning capabilities of our approach properly. An overview of each task is illustrated in Figure 3.

### A.3 Baselines

To evaluate C-MCTD, we compare its performance against diverse baselines that combine different extending methods for connecting plans with various diffusion planners. We describe these components below.

#### A.3.1 Extending methods

**Replan.** This basic method generates plans when the plan generated from diffusion planner is limited to the trajectory length seen during training. By iteratively planning from the final state of each previous plan, it enables extendable planning capabilities. However, since each individual planning step is constrained to short-term horizons, this approach typically yields sub-optimal solutions.

**DataStitch [3, 19].** An alternative approach to planning beyond the length of training trajectories involves extending the trajectories in the dataset itself. Since all data points in the dataset are not connectable, methods such as Chen et al. [3] and Li et al. [19] train separate diffusion planners and inverse dynamics models to generate simulated trajectories that connect existing data. Notably, Chen et al. [3] performed this process multiple times to extend trajectories up to 7. Following this approach, we extend the trajectories in the *stitch* dataset to 5 times their original length and train our models on these extended datasets.

#### A.3.2 Diffusion planners

**Diffuser [15].** As a primary diffusion planning baseline, we include Diffuser, which pioneered the approach of training diffusion models on state-action pairs concatenated as sequence data—a format

now widely adopted by subsequent diffusion planning methods. We evaluate Diffuser with both the Replan extending method and the DataStitch extending method by training it on the extended datasets.

**Diffusion Forcing [2].** This variant of Diffuser tokenizes the data and applies varying noise levels during the denoising process, enabling causal denoising that prioritizes near-future states. Of the two versions proposed in Chen et al. [2], one using recurrent neural networks and another using Transformer backbones, we employ the Transformer-based implementation. We evaluate Diffusion Forcing with both the Replan and DataStitch extending methods (note that since Diffusion Forcing was originally designed for replanning, we do not separately label the "Replan" variant).

**Sub-trajectory Stitching with Diffusion (SSD) [16].** Kim et al. [16] introduced a diffusion planning method that achieves extendability through a learned value function. Even when trained on short trajectories, the learned value function guides the model to generate more task-relevant plans through concatenation. While this approach addresses the implementation of extendable planning, we empirically found that value function learning becomes challenging when data quality is suboptimal, whereas inference-time scaling provides a more robust solution. Since SSD was originally designed for replanning, so we didn't label "Replan" as done for Diffusion Forcing. And it already addresses extendable planning, we did not evaluate it with extended datasets.

**Compositional Diffuser (CompDiffuser) [22].** Luo et al. [22] proposed another diffusion planning method for extendable planning. Their approach divides training trajectories into sub-trajectories and trains a diffusion planner to generate sub-trajectories with nearby context. Through this framework, CompDiffuser can generalize to simultaneously generate multiple nearby sub-trajectories, enabling plan generation beyond the trajectory lengths observed during training. Similar to SSD, CompDiffuser was designed for replanning, so we did not label "Replan" and we did not apply extended datasets to this baseline.

**Monte Carlo Tree Diffusion (MCTD) [30].** Yoon et al. [30] introduced a framework that implements inference-time scaling in the diffusion denoising process. Rather than merely increasing denoising steps, MCTD branches possible solutions to explore alternative paths for finding optimal solutions. Guided by a reward function, it demonstrates superior performance across diverse planning tasks compared to previous approaches, showcasing the potential of diffusion planners for complex planning tasks. We evaluated MCTD with both the Replan and DataStitch extending methods.

## A.4 Model hyperparameters

Many hyperparameters follow the settings in the previous work [30], but the additional hyperparameters such as the employed diffusion planners are discussed in evaluation details for each environment. For SSD, we applied the default configurations set in their source codes, <https://github.com/r1atjddb/SSD>.

## A.5 Evaluation details

This section outlines the specific configurations used for each benchmark evaluation.

### A.5.1 Long-horizon maze environments

For long-horizon maze environments, we trained our models using the *stitch* datasets to evaluate extendable planning capabilities, with one exception: the value learning policy [29]. We observed that this policy could not be effectively trained with the stitched datasets due to data quality limitations. Since the quality of the value learning policy falls outside our evaluation scope, we employed the *navigate* dataset for training this particular policy. Across all maze environments, we used a trajectory length of 100 during training, while the maximum length in the dataset was 200, allowing us to train with more diverse data patterns.

Additionally, we implemented a best-of- $N$  search-based diffusion planner [32] built upon Diffusion Forcing [2]. Best-of- $N$  represents an inference-time scaling method for diffusion models that generates  $N$  candidate plans and selects the top sample according to a specified reward function. We

chose this approach because the 100-step length is sufficiently short to be effectively searched using best-of- $N$  without requiring MCTD [30]. We used the same reward function as specified in previous work [30] and set  $N = 50$  for our experiments.

For Distributed Composer (DC) and Preplan Composer (PC), we generated random positions from the training datasets using clustering algorithms [20, 24]. Specifically, we employed 10, 30, and 70 clustering centers for medium, large, and giant-sized maps, respectively. For both DC and PC, we limited our search to just 10 iterations with concatenation of up to 3 plans, a significantly less intensive search compared to the Online Composer, which used up to 500 searches and concatenation of up to 10 plans.

### A.5.2 Robot arm manipulation environment

For the robot arm manipulation environment, where each object movement plan can exceed 100 steps and involve greater complexity, we employed MCTD [30] as our diffusion planner.

Distinctively for this environment, we implemented a novel strategy called plan caching, which stores previously generated plans for object movements and reuses them compositionally without regeneration. In this approach, C-MCTD caches plans with associated metadata: target object label, start position, and goal position. When a future task requires a plan matching these key parameters, C-MCTD utilizes the cached plan rather than generating a new one, enhancing computational efficiency.

### A.5.3 Visual maze environment

We followed the MCTD [30] setup for the visual maze environment. Observations and targets are 64×64 RGB images, encoded into an 8-dimensional latent space using a pretrained Variational Autoencoder (VAE) [17]. Inverse dynamics and position estimation are handled by a simple, pretrained multi-layer perceptron (MLP) trained on offline data. For DC and PC, we applied clustering to the estimated positions, selecting 3 and 5 representative points for the medium and large maps, respectively.

## B Additional experimental results

### B.1 Plan Caching ablations Study

Table 5: **Robot arm cube manipulation run time comparison.** Run time (sec.) across increasing complexity manipulation tasks involving single, double, triple, and quadruple cube arrangements in OGBench, presented as mean  $\pm$  standard deviation.

Method	Success Rate (%) $\uparrow$			
	single	double	triple	quadruple
OnlineComposer	<b>19.4</b> $\pm 0.3$	50.5 $\pm 1.4$	308.4 $\pm 33.5$	1432.8 $\pm 300.1$
OnlineComposer with Plan Cache	<b>19.4</b> $\pm 0.2$	<b>45.5</b> $\pm 0.7$	<b>166.9</b> $\pm 27.1$	<b>242.8</b> $\pm 16.1$

We conducted an ablation study to evaluate the impact of our plan caching approach, with results presented in Table 5. The findings demonstrate that for simpler manipulation tasks involving single or double cubes, the runtime difference between standard Online Composer and Online Composer with Plan Cache is relatively minimal. However, as task complexity increases with triple and quadruple cube arrangements, the performance gap widens exponentially.

This exponential difference occurs because higher object counts dramatically increase the combinatorial complexity of object-wise plans. In the quadruple cube scenario, Online Composer with Plan Cache achieves approximately a 6 $\times$  runtime reduction (242.8s versus 1432.8s) compared to the standard implementation. These results clearly demonstrate the efficiency benefits of plan caching, particularly for complex manipulation tasks where multiple object interactions create a combinatorial explosion in the planning space.

**Algorithm 2** Online Composer

---

```

1: procedure ONLINE COMPOSER( $s_{\text{start}}, s_{\text{goal}}, D, H, L, B, \mathcal{G}$ )
2:    $v_{\text{root}} \leftarrow \text{Node}(s_{\text{start}}, \text{null})$   $\triangleright$  Create root node with start state
3:    $\mathcal{T} \leftarrow (V = \{v_{\text{root}}\}, E = \emptyset)$   $\triangleright$  Initialize tree with root node
4:    $\text{expandedNodes} \leftarrow 0$ 
5:   while  $\text{expandedNodes} < B$  and no node close to  $s_{\text{goal}}$  do
6:      $v \leftarrow \text{SelectNodeToExpand}(\mathcal{T})$   $\triangleright$  Based on UCT value
7:      $G_s \leftarrow \text{SelectGuidanceSet}(v, \mathcal{G})$   $\triangleright$  Select guidance set
8:      $\tau_{\text{sub}} \leftarrow D(v.\text{plan}, s_{\text{goal}}, H, G_s)$   $\triangleright$  Generate guided sub-plan
9:      $\tau_{\text{child}} \leftarrow v.\text{plan} \oplus \tau_{\text{sub}}$ 
10:     $\text{reward} \leftarrow \text{FastReplanningSimulation}(\tau_{\text{child}}, s_{\text{goal}}, D, H, L)$ 
11:     $v_{\text{child}} \leftarrow \text{Node}(\tau_{\text{child}}, \text{reward})$ 
12:     $\mathcal{T}.V \leftarrow \mathcal{T}.V \cup \{v_{\text{child}}\}$   $\triangleright$  Add child node to tree
13:     $\mathcal{T}.E \leftarrow \mathcal{T}.E \cup \{(v, v_{\text{child}})\}$   $\triangleright$  Add edge to tree
14:     $\text{Backpropagate}(v, \text{reward})$   $\triangleright$  Update ancestors' values
15:     $\text{expandedNodes} \leftarrow \text{expandedNodes} + 1$ 
16:  end while
17:  if node  $v_{\text{goal}}$  exists with  $\text{dist}(v_{\text{goal}}.\text{plan}, s_{\text{goal}}) \leq \varepsilon$  then
18:    return  $v_{\text{goal}}.\text{plan}$ 
19:  else
20:    return failure
21:  end if
22: end procedure
23: procedure FASTREPLANNINGSIMULATION( $\tau_{\text{current}}, s_{\text{goal}}, D, H, L$ )
24:    $\tau_{\text{remaining}} \leftarrow \emptyset$   $\triangleright$  Initialize empty trajectory
25:    $s_{\text{current}} \leftarrow \tau_{\text{current}}[-1]$ 
26:   while  $\text{length}(\tau_{\text{remaining}}) < L$  do
27:      $\tau_{\text{fast}} \leftarrow D.\text{FastDenoise}(s_{\text{current}}, s_{\text{goal}}, H)$   $\triangleright$  Fast denoising
28:      $\tau_{\text{remaining}} \leftarrow \tau_{\text{remaining}} \oplus \tau_{\text{fast}}$   $\triangleright$  Append trajectory
29:      $s_{\text{current}} \leftarrow \tau_{\text{fast}}[H]$   $\triangleright$  Update current state
30:     if no progress or iteration limit reached then
31:       break
32:     end if
33:   end while
34:   return  $\text{EvaluateReward}(\tau_{\text{remaining}})$   $\triangleright$  Return estimated reward
35: end procedure

```

---

---

**Algorithm 3** Distributed Composer (DC)

---

**Require:**  $s_{\text{start}}$ : start state,  $s_{\text{goal}}$ : goal state,  $D$ : diffusion planner,  $H$ : sub-plan horizon,  $L$ : full-plan horizon,  $B$ : expansion budget,  $N$ : number of origin positions

**Ensure:** A full trajectory  $\tau_{\text{full}}$  from  $s_{\text{start}}$  to  $s_{\text{goal}}$  or failure

```
1: procedure DISTRIBUTED_COMPOSER( $s_{\text{start}}, s_{\text{goal}}, D, H, L, B, N$ )
2:    $\mathcal{S} \leftarrow \{s_{\text{start}}\} \cup \text{SamplePositions}(N - 1)$   $\triangleright N$  origin positions
3:    $G \leftarrow (\mathcal{S}, \emptyset)$   $\triangleright$  Initialize connectivity graph
4:   expandedTrees  $\leftarrow 0$ 
5:   while expandedTrees  $< B$  do
6:      $s_i \leftarrow \text{SelectOriginPosition}(\mathcal{S}, G)$   $\triangleright$  Tree expansion can be implemented in parallel
7:      $\tilde{p}_\theta(\mathbf{x}|s_i) \propto p_\theta(\mathbf{x}|s_i) \exp(\mathcal{J}_\phi(\mathbf{x}))$   $\triangleright$  Define guided distribution
8:      $\mathcal{T}_i \leftarrow \text{GrowTree}(s_i, \tilde{p}_\theta, D, H, L)$   $\triangleright$  Grow tree using Online Composer
9:     for each  $s_j \in \mathcal{S} \setminus \{s_i\}$  do
10:      for each node  $v \in \mathcal{T}_i$  do
11:        if  $\min_{p \in v.\text{plan}} \text{dist}(p, s_j) < \epsilon$  then
12:           $G.E \leftarrow G.E \cup \{(s_i, s_j, v.\text{plan})\}$   $\triangleright$  Add connection to graph
13:          break
14:        end if
15:      end for
16:    end for
17:    expandedTrees  $\leftarrow$  expandedTrees + 1
18:    if path exists from  $s_{\text{start}}$  to  $s_{\text{goal}}$  in  $G$  then
19:      path  $\leftarrow \text{ShortestPath}(G, s_{\text{start}}, s_{\text{goal}})$   $\triangleright$  Dijkstra's or A*
20:       $\tau_{\text{full}} \leftarrow \text{StitchPath}(\text{path})$   $\triangleright$  Stitch plans along path
21:      if length( $\tau_{\text{full}}$ )  $\leq L$  then
22:        return  $\tau_{\text{full}}$ 
23:      end if
24:    end if
25:  end while
26:  return failure
27: end procedure
28: procedure GROWTREE( $s_i, \tilde{p}_\theta, D, H, L$ )
29:    $v_{\text{root}} \leftarrow \text{Node}(s_i, \emptyset)$   $\triangleright$  Create root node
30:    $\mathcal{T} \leftarrow (V = \{v_{\text{root}}\}, E = \emptyset)$   $\triangleright$  Initialize tree
31:   for  $k = 1$  to  $K$  do  $\triangleright K$ : tree expansion iterations
32:      $v \leftarrow \text{SelectNodeToExpand}(\mathcal{T})$ 
33:      $\tau_{\text{sub}} \leftarrow D(v.\text{plan}, \tilde{p}_\theta, H)$   $\triangleright$  Sample from guided distribution
34:      $\tau_{\text{child}} \leftarrow v.\text{plan} \oplus \tau_{\text{sub}}$ 
35:     reward  $\leftarrow \text{FastReplanningSimulation}(\tau_{\text{child}}, s_{\text{goal}}, D, H, L)$ 
36:      $v_{\text{child}} \leftarrow \text{Node}(\tau_{\text{child}}, \text{reward})$ 
37:      $\mathcal{T}.V \leftarrow \mathcal{T}.V \cup \{v_{\text{child}}\}$ 
38:      $\mathcal{T}.E \leftarrow \mathcal{T}.E \cup \{(v, v_{\text{child}})\}$ 
39:   end for
40:   return  $\mathcal{T}$ 
41: end procedure
```

---

---

**Algorithm 4** Preplan Composer (PC) - Building Plan Graph

---

**Require:**  $\mathcal{E}$ : environment,  $D$ : diffusion planner,  $H$ : sub-plan horizon,  $K$ : number of waypoints

**Ensure:** Cognitive map  $\mathcal{M}$  encoding environment connectivity

```
1: procedure BUILDPLANGRAPH( $\mathcal{E}, D, H, K$ )                                ▷ Pre-building phase
2:    $\mathcal{W} \leftarrow \text{SelectWaypoints}(\mathcal{E}, K)$                                 ▷ Identify  $K$  strategic waypoints
3:    $\mathcal{M} \leftarrow (\mathcal{W}, \emptyset)$                                           ▷ Initialize plan graph
4:   for each pair  $(w_i, w_j) \in \mathcal{W} \times \mathcal{W}$  where  $i \neq j$  do
5:      $\tilde{p}_\theta(\mathbf{x}|w_i) \propto p_\theta(\mathbf{x}|w_i) \exp(-\text{dist}(\mathbf{x}, w_j))$         ▷ Position-based guidance
6:      $\tau_{ij} \leftarrow \text{OC}(w_i, w_j, D, H, \tilde{p}_\theta)$                         ▷ Generate plan
7:     if  $\tau_{ij} \neq \emptyset$  then                                          ▷ If connection successful
8:        $r_{ij} \leftarrow \text{ComputeCost}(\tau_{ij})$                             ▷ Assign cost to connection
9:        $\mathcal{M.C} \leftarrow \mathcal{M.C} \cup \{(w_i, w_j, \tau_{ij}, r_{ij})\}$         ▷ Add to map
10:    end if
11:  end for
12:  return  $\mathcal{M}$                                                          ▷ Return completed cognitive map
13: end procedure
```

---

---

**Algorithm 5** Preplan Composer - Inference

---

**Require:**  $s_{\text{start}}$ : start state,  $s_{\text{goal}}$ : goal state,  $\mathcal{M}$ : cognitive map,  $D$ : diffusion planner,  $H$ : sub-plan horizon,  $L$ : full-plan horizon

**Ensure:** A full trajectory  $\tau_{\text{full}}$  from  $s_{\text{start}}$  to  $s_{\text{goal}}$  or failure

```
1: procedure PC-INFERENC( $s_{\text{start}}, s_{\text{goal}}, \mathcal{M}, D, H, L$ )
2:    $\mathcal{W}_{\text{near}}(s_{\text{start}}) \leftarrow \text{FindNearWaypoints}(s_{\text{start}}, \mathcal{M.W})$ 
3:    $\mathcal{W}_{\text{near}}(s_{\text{goal}}) \leftarrow \text{FindNearWaypoints}(s_{\text{goal}}, \mathcal{M.W})$ 
4:    $\mathcal{C}_{\text{start}} \leftarrow \emptyset$ 
5:   for each  $w_i \in \mathcal{W}_{\text{near}}(s_{\text{start}})$  do
6:      $\tau_{s \rightarrow i} \leftarrow \text{OC}(s_{\text{start}}, w_i, D, H, L)$ 
7:     if  $\tau_{s \rightarrow i} \neq \emptyset$  then
8:        $r_{s \rightarrow i} \leftarrow \text{ComputeCost}(\tau_{s \rightarrow i})$ 
9:        $\mathcal{C}_{\text{start}} \leftarrow \mathcal{C}_{\text{start}} \cup \{(s_{\text{start}}, w_i, \tau_{s \rightarrow i}, r_{s \rightarrow i})\}$ 
10:    end if
11:  end for
12:   $\mathcal{C}_{\text{goal}} \leftarrow \emptyset$ 
13:  for each  $w_j \in \mathcal{W}_{\text{near}}(s_{\text{goal}})$  do
14:     $\tau_{j \rightarrow g} \leftarrow \text{OC}(w_j, s_{\text{goal}}, D, H, L)$ 
15:    if  $\tau_{j \rightarrow g} \neq \emptyset$  then
16:       $r_{j \rightarrow g} \leftarrow \text{ComputeCost}(\tau_{j \rightarrow g})$ 
17:       $\mathcal{C}_{\text{goal}} \leftarrow \mathcal{C}_{\text{goal}} \cup \{(w_j, s_{\text{goal}}, \tau_{j \rightarrow g}, r_{j \rightarrow g})\}$ 
18:    end if
19:  end for
20:   $G_{\text{aug}} \leftarrow (\mathcal{M.W} \cup \{s_{\text{start}}, s_{\text{goal}}\}, \mathcal{M.C} \cup \mathcal{C}_{\text{start}} \cup \mathcal{C}_{\text{goal}})$ 
21:   $\text{path} \leftarrow \text{ShortestPath}(G_{\text{aug}}, s_{\text{start}}, s_{\text{goal}})$ 
22:  if  $\text{path} \neq \emptyset$  then
23:     $\tau_{\text{full}} \leftarrow \text{StitchPath}(\text{path})$                                 ▷ Compose final solution
24:    if  $\text{length}(\tau_{\text{full}}) \leq L$  then
25:      return  $\tau_{\text{full}}$ 
26:    end if
27:  end if
28:  return failure
29: end procedure
```

---