

A Experiment Details

A.1 Computation Resources

The experiments were conducted on a computing server equipped with 8 NVIDIA RTX 4090 GPUs, 512GB system memory, and a 96-thread CPU. Model training required approximately 3-6 hours per model, while inference for each experimental evaluation took approximately up to 5 minutes.

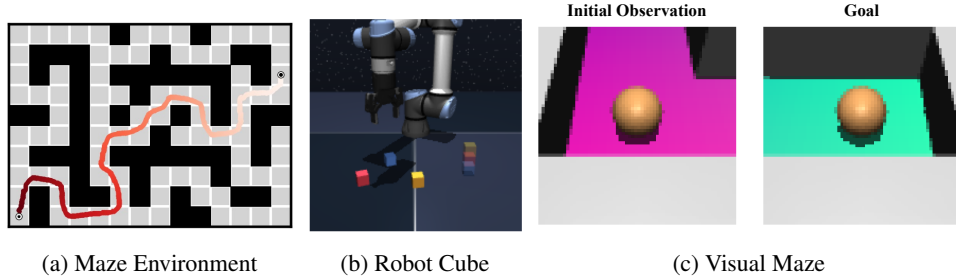


Figure 5: **The task illustrations:** (a) long-horizon maze navigation, (b) multi-object robotic manipulation requiring compositional planning, and (c) visual maze planning from high-dimensional raw RGB observations.

A.2 Environment Details

Following prior work [31], we evaluated our methods (P-MCTD, S-MCTD and Fast-MCTD) on challenging scenarios from the goal-conditioned task benchmark, OGBench [21], including long-horizon maze navigation, robot arm manipulation, and visually observable maze environments as illustrated in Figure 4. All evaluation tasks were unseen during training, requiring the models to generalize their learned planning capabilities to novel scenarios at inference time.

A.2.1 Long-Horizon Maze

To assess performance on extensive planning horizons, we tested our methods on point-mass and ant robot navigation tasks across three increasingly complex environments: medium, large, and giant mazes. Models were trained on the *navigate dataset* containing long trajectories but without specific task knowledge. At inference time, models must leverage their learned capabilities to solve previously unseen maze configurations, testing their generalization to complex long-horizon planning problems.

A.2.2 Robot Arm Manipulation

The robot manipulation tasks require compositional planning to move multiple objects in specific sequences. For example, when tasked with stacking objects in a predetermined order, the model must generate a coherent plan that respects these sequential constraints. As demonstrated in prior work [31], holistic diffusion-based planners often struggle with such compositional tasks, particularly without explicit search or reward guidance. We evaluated Fast-MCTD in this challenging domain, focusing on efficiency improvements within a replanning framework while maintaining MCTD’s strong performance on compositional tasks.

A.2.3 Visual Maze

To evaluate MCTD in high-dimensional, partially observable settings, previous work [31] utilized the Visual Antmaze task from OGBench [21]. In this task, the agent must navigate from an initial observation to a target, both represented as 64x64 RGB images with 3 color channels. The images are encoded into an 8-dimensional latent space using a Variational Autoencoder (VAE) [13], providing a compact representation for efficient planning.

A.3 Baselines

We compared Fast-MCTD against several diffusion-based planning approaches:

- **Diffuser** [12]: We implemented the standard Diffuser with classifier-guided generation [8] as our foundational baseline [31], providing comparable guidance mechanisms across all tested methods.
- **Diffuser-Replan**: To strengthen this baseline, we extended Diffuser with periodic replanning capabilities. This variant regenerates complete plans at fixed intervals, allowing it to adjust to intermediate state observations and potentially improve performance through iterative refinement.
- **Diffusion Forcing** [2]: This method represents a variant of diffusion planning that incorporates causal noise scheduling with tokenized trajectories but without explicit search mechanisms. Its default implementation includes replanning capabilities, making it a strong comparative baseline alongside MCTD.
- **MCTD** [31]: As our primary baseline, MCTD represents the state-of-the-art in inference-time scaling for diffusion planning. It combines causal noise scheduling with tree-based search, demonstrating exceptional performance across diverse planning tasks. However, our investigation reveals significant computational inefficiencies, particularly in scenarios where its planning capabilities are most valuable.

A.4 Model Hyperparameters

While maintaining consistency with the hyperparameter configurations used in prior work [31], but we describe the detailed hyperparameters to improve the reproducibility. We highlight the key parameters specific to our methods while complete implementation details will be available.

Table 7: Diffuser Hyperparameters

Hyperparameter	Value
Learning Rate	$2e - 4$
EMA Decay	0.995
Precision in Training/Inference	32
Batch Size	32
Max Training Steps	20000
Planning Horizon	Task dependent, discussed in Section A.5
Open Loop Horizon	50 for Diffuser-Replan, otherwise planning horizon
Guidance Scale	0.1
Beta Schedule	Cosine
Diffusion Model Objective	x_0 -prediction
U-Net Depth	4
Kernel Size	5
The Number of Channels	32, 128, 256

Table 8: Value-Learning Policy Hyperparameters

Hyperparameter	Value
Learning Rate	$3e - 4$
Learning Eta	1.0
Max Q Backup	False
Reward Tune	cql_antmaze
The Number of Training Epochs	2000
Gradient Clipping	7.0
Top-k	1
Target Steps	10
Randomness on Data Sampling (p)	0.2

Table 9: Diffusion Forcing Hyperparameters

Hyperparameter	Value
Learning Rate	$5e - 4$
Weight Decay	$1e - 4$
Warmup Steps	10000
Precision in Training	16-mixed
Precision in Inference	32
Batch Size	1024
Max Training Steps	200005
The Number of Frame Stack	10
Planning Horizon	Task dependent, discussed in Section A.5
Open Loop Horizon	50
Causal Mask	Not Used
Guidance Scale	3 for medium mazes and 2 for other mazes
Scheduling Matrix	pyramid
Stabilization Level	10
Beta Schedule	Linear
Diffusion Model Objective	x_0 -prediction
DDIM Sampling eta	0.0
Network Size	128
The Number of Layers	12
The Number of Attention Heads	4
The Feedforward Network Dimension	512

A.5 The Evaluation Details

Similar to the hyperparameters, many details are consistent with the settings established in prior work [31], while describing key experimental settings below.

A.5.1 Maze Navigation with Point-Mass and Ant Robots

Guidance Set For pointmaze tasks, $[0, 0.1, 0.5, 1, 2]$ is applied and $[0, 1, 2, 3, 4, 5]$ is set in antmaze evaluation.

Planning Horizon 500 planning horizon is applied for medium and large sized mazes and 1000 for giant sized mazes.

Low-level Controller For pointmaze tasks, the heuristic controller [12] is applied and the diffusion-based value-learning policy [28] is utilized for antmaze [4]. For value-learning policy low-level controller acts to the given subgoal which is re-assigned after achieving with 10 steps interval.

A.5.2 Robot Arm Cube Manipulation

Guidance Set $[1, 2, 4]$ is applied for each object. MCTDs applies the object-wise guidance, so the size of guidance set is increased as the number of objects increases in the environment.

Planning Horizon 200 for single cube tasks and 500 for other tasks.

Low-level Control The value-learning policy [28] is applied as done in antmaze tasks.

A.5.3 Visual Pointmaze

Guidance Set. For visual maze tasks, a guidance scale set of $[0, 0.1, 0.5, 1, 2]$ is applied.

Planning Horizon. A fixed planning horizon of 500 steps is used for medium and large-sized mazes.

Low-level Controller. Inverse Dynamics (InvDyna) models were trained following the MCTD framework [31] using offline datasets encoded by a pretrained VAE. These models predict the action \hat{a}_t required to transition from the current state s_t to the next state s_{t+1} , given the partially observable nature of the task. To capture velocity-related information, the InvDyna model conditions its predictions on both the previous state x_{t-1} and the current state s_t , formalized as:

Table 10: MCTDs Hyperparameters

Hyperparameter	Value
Learning Rate	$5e - 4$
Weight Decay	$1e - 4$
Warmup Steps	10000
Precision in Training	16-mixed
Precision in Inference	32
Batch Size	1024
Max Training Steps	200005
The Number of Frame Stack	10
Planning Horizon	Task dependent, discussed in Section A.5
Open Loop Horizon	50 for Replan otherwise Planning Horizon
Causal Mask	Not Used
Scheduling Matrix	pyramid
The Maximum Number of Search	500
Guidance Set	Task dependent, discussed in Section A.5
The number of Partial Denoising	20
The Jumpy Denoising Interval	10
Stabilization Level	10
Beta Schedule	Linear
Diffusion Model Objective	x_0 -prediction
DDIM Sampling eta	0.0
Network Size	128
The Number of Layers	12
The Number of Attention Heads	4
The Feedforward Network Dimension	512
Parallelism Degree	200 for P-MCTD and Fast-MCTD, otherwise 1
Parallel Search Visitation Count Weight w	1.0
Subsampling Interval H	5 for S-MCTD and Fast-MCTD, otherwise 1

$$\hat{a}_t = \text{InvDyna}((s_{t-1}, s_t), s_{t+1}) \quad (7)$$

Guidance in POMDP. To enhance spatial awareness in partially observable environments, a Position Estimator, pretrained on offline datasets, predicts the agent’s location in the VAE’s latent space. This module provides a coarse but informative positional signal.

B Additional Experimental Results

B.1 Leaf Parallelization Ablations

Table 11: **Leaf Parallelization Ablation Study Results.** Success rates and planning times (\pm std) across PointMaze environment on medium, large giant sized map for Leaf Parallelization ablation.

Env.	Method	Success Rate \uparrow (%)			Planning Time \downarrow (sec.)		
		medium	large	giant	medium	large	giant
PointMaze	No Leaf Par. (P-MCTD)	100 ± 0	100 ± 0	100 ± 0	8.5 ± 1.6	7.4 ± 1.3	12.8 ± 2.5
	Leaf Par.	98 ± 6	100 ± 0	100 ± 0	7.4 ± 0.9	6.7 ± 0.6	17.0 ± 2.8

While P-MCTD implements parallel search through delayed tree updates and search-aware selection, alternative parallelization strategies exist for MCTS [18]. One such approach is leaf parallelization, which expands multiple children from a single selected node in parallel. This technique potentially reduces selection overhead by processing multiple expansions per selection operation.

As shown in Table 11, leaf parallelization yields mixed results across different environment scales. In medium and large mazes, it shows modest efficiency improvements (7.4s vs. 8.5s and 6.7s vs. 7.4s, respectively) with minimal impact on success rates. However, in the giant maze environment, leaf parallelization significantly increases planning time (17.0s vs. 12.8s), demonstrating reduced efficiency precisely where computational optimization is most critical.

878 This performance degradation in complex environments likely stems from leaf parallelization’s
 879 interference with the exploration-exploitation balance maintained by our search-aware selection
 880 mechanism, Equation 5. By forcing expansion of multiple children from the same node, leaf paral-
 881 lelization may concentrate computational resources on potentially suboptimal branches, particularly
 882 detrimental in larger search spaces where strategic node selection is crucial. These results reinforce
 883 our approach of parallelizing across different branches of the search tree rather than within individual
 884 branches, especially for complex long-horizon planning scenarios.

885 B.2 Distillation of Diffusion Planner Ablations

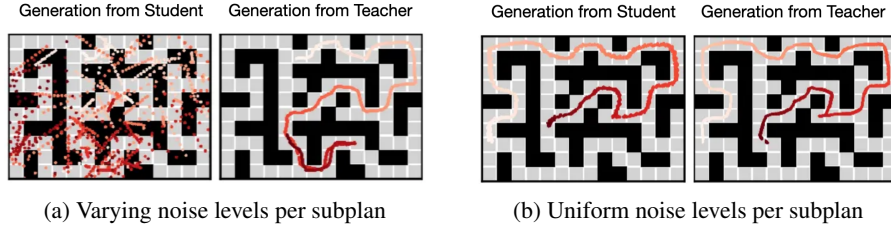


Figure 6: **Distillation challenges in MCTD.** Trajectory quality from distilled diffusion models varies significantly based on noise level distribution at training. (a) Using variable noise levels (required for causal scheduling) leads to poor trajectory quality. (b) Using uniform noise levels produces cleaner trajectories but is incompatible with MCTD’s causal scheduling requirements.

886 Diffusion model distillation represents a promising alternative approach for optimizing denoising
 887 processes by training student models to predict the results of multiple denoising steps performed by a
 888 teacher model [22]. We conducted a preliminary investigation of this technique as another potential
 889 direction for optimizing MCTD.

890 Following the progressive distillation approach [22], we trained a student model to predict two-step
 891 denoised outputs from the teacher model. However, our experiments revealed significant challenges
 892 in trajectory quality, as illustrated in Figure 6. Upon analysis, we identified that MCTD’s causal
 893 noise scheduling—which applies different noise levels to different subplans—introduces unique
 894 complications for distillation approaches. While standard diffusion distillation typically assumes
 895 uniform noise levels across the entire generated content, MCTD’s varying per-subplan noise levels
 896 create a substantially more complex learning task for the student model.

897 Our findings suggest that effective distillation for MCTD would require specialized techniques
 898 capable of handling these heterogeneous noise distributions. This represents an intriguing avenue for
 899 future research, but we leave this as one of the future works.

Algorithm 2 Fast Monte Carlo Tree Diffusion

```

1: procedure FAST-MCTD(root, iterations, parallelism_degree, jump_interval)
2:   for i = 1 to iterations by parallelism_degree do
3:     nodes_to_expand  $\leftarrow$  []
4:     temp_visit_counts  $\leftarrow$  {} ▷ Track parallel selections
5:     for j = 1 to parallelism_degree do ▷ Search-aware selection
6:       node  $\leftarrow$  root
7:       while ISFULLYEXPANDED(node) and not ISLEAF(node) do
8:         node  $\leftarrow$  SEARCHAWAREUCT(node, temp_visit_counts)
9:         temp_visit_counts[node]  $\leftarrow$  temp_visit_counts.get(node, 0) + 1
10:      end while
11:      APPEND(nodes_to_expand, node)
12:    end for
13:    expansions  $\leftarrow$  [] ▷ Parallel expansion phase
14:    for node  $\in$  nodes_to_expand do
15:      if ISEXPANDABLE(node) then
16:        gs  $\leftarrow$  SELECTMETACTION(node)
17:        APPEND(expansions, (node, gs))
18:      end if
19:    end for
20:    new_children  $\leftarrow$  BATCHDENOISESUBPLANS(expansions, jump_interval)
21:    for (node, child)  $\in$  new_children do
22:      ADDCHILD(node, child)
23:    end for
24:    simulations  $\leftarrow$  [] ▷ Parallel simulation phase
25:    for (node, child)  $\in$  new_children do
26:      partial  $\leftarrow$  GETPARTIALTRAJECTORY(child)
27:      APPEND(simulations, (node, child, partial))
28:    end for
29:    rewards  $\leftarrow$  BATCHFASTSPARSEDENOISING(simulations, jump_interval)
30:    for (node, child, reward)  $\in$  rewards do ▷ Delayed tree update
31:      current  $\leftarrow$  child
32:      while current  $\neq$  null do
33:        current.visitCount  $\leftarrow$  current.visitCount + 1
34:        current.value  $\leftarrow$  MAX(current.value, reward)
35:        UPDATEMETACTIONSCHEDULE(current, reward)
36:        current  $\leftarrow$  current.parent
37:      end while
38:    end for
39:  end for
40:  return BESTCHILD(root)
41: end procedure

```
