Train-Once Plan-Anywhere Kinodynamic Motion Planning via Diffusion Trees

Yaniv Hassidof[*] **Tom Jurgenson** **Kiril Solovey**
Viterbi Faculty of Electrical and Computer Engineering
Technion–Israel Institute of Technology, Haifa, Israel

Abstract: Kinodynamic motion planning is concerned with computing collision-free trajectories while abiding by the robot's dynamic constraints. This critical problem is often tackled using sampling-based planners (SBPs) that explore the robot's high-dimensional state space by constructing a search tree via action propagations. Although SBPs can offer global guarantees on completeness and solution quality, their performance is often hindered by slow exploration due to uninformed action sampling. Learning-based approaches can yield significantly faster runtimes, yet they fail to generalize to out-of-distribution (OOD) scenarios and lack critical guarantees, e.g., safety, thus limiting their deployment on physical robots. We present Diffusion Tree (DiTree): a *provably-generalizable* framework leveraging diffusion policies (DPs) as informed samplers to efficiently guide state-space search within SBPs. DiTree combines DP's ability to model complex distributions of expert trajectories, conditioned on local observations, with the completeness of SBPs to yield *provably-safe* solutions within a few action propagation iterations for complex dynamical systems. We demonstrate DiTree's power with an implementation combining the popular RRT planner with a DP action sampler trained on a *single environment*. In comprehensive evaluations on OOD scenarios, DiTree is on average 3x faster than classical SBPs, and outperforms all other approaches by achieving roughly 30% higher success rate. Project webpage: sites.google.com/view/ditree.

Keywords: Kinodynamic motion planning; diffusion models; nonlinear systems

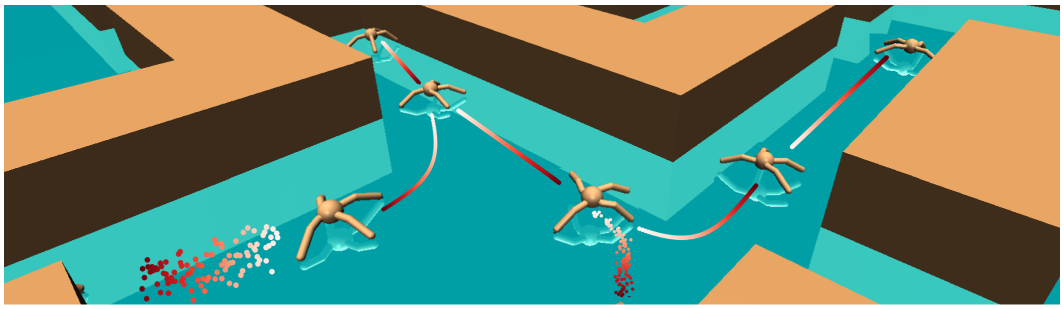


Figure 1: Visualization of DiTree on D4RL's AntMaze setting in a random maze *unseen* during training. Edges are generated via diffusion.

1 Introduction

Robotic mobility has been a longstanding goal in artificial intelligence, pivotal for real-world applications such as autonomous driving [1], drones [2], and humanoid robots [3]. Such systems are often underactuated [4], requiring motion planners to account not only for environmental constraints but also for the physical limitations and dynamics of the robot itself, in a problem called *kinodynamic motion planning* (KMP) [5]. Despite its importance, KMP remains challenging due to the complexity of searching high-dimensional state and action spaces with non-linear dynamics. Echoing

[*]Corresponding author; Email: yaniv_hass@campus.technion.ac.il, tomj@campus.technion.ac.il, kirilsol@technion.ac.il.

Sutton's [6] "Bitter Lesson", formidable challenges in Computer Science are often solved by large-scale general-purpose methods rather than specialized heuristics, with *search* and *learning* emerging as two approaches capable of arbitrary scaling, rendering them essential in tackling KMP.

A cornerstone of search approaches for KMP are sampling-based planners (SBPs) [7]. By sampling random actions and simulating them through the dynamic model, SBPs grow a collision-free tree that spans the state space. SBPs are valued for their simplicity, versatility, and strong theoretical foundations. However, their exhaustive exploration can be inefficient in high-dimensional spaces, particularly in environments densely cluttered with obstacles or involving complex dynamics.

Learning-based approaches have emerged as a promising alternative for improving motion planning efficiency. Prominent among them, diffusion planning [8] casts planning as a form of probabilistic inference [9] and samples from the multi-modal distribution of expert behaviors to synthesize diverse and high-quality trajectories. Unfortunately, such models struggle to generalize to unseen environments, and inherently lack guarantees such as collision avoidance.

Contribution. Motivated by breakthroughs combining learning and search, e.g., AlphaGo [10, 11] and LLM reasoning [12, 13], we propose a learning-meets-search strategy for KMP. We introduce *Diffusion Tree (DiTree)*, a novel framework that combines diffusion models (DMs) with sampling-based tree search for efficient KMP. DiTree leverages DMs as powerful, environment-aware motion priors to guide exploration toward promising trajectories. By integrating these priors with SBPs, DiTree ensures collision avoidance and dynamic feasibility, while successfully navigating unseen environments. We demonstrate that DiTree achieves substantial efficiency improvements and outperforms state-of-the-art methods across diverse tasks and robotic platforms. We explore the set of unique challenges and trade-offs entailed with employing a DM for tree search in our ablation studies.

Organization. We first survey related work (Sec. 2), provide background on KMP, SBPs, and DMs (Sec. 3), and then introduce our DiTree framework (Sec. 4). We then present implementation details and practical considerations, followed by a comprehensive evaluation (Sec. 5). Sec. 6 concludes with a discussion and future work, followed by limitations in Sec. 7.

2 Related Work

Sampling-based planners [5] fall broadly into two categories: *geometric* and *kinodynamic* planners. Geometric SBPs [14, 15, 16] construct a graph by sampling states in the configuration space and connecting them with straight edges. While conceptually simple and widely used, they assume the robot can move directly between any two states without regard for its dynamics. *Kinodynamic* SBPs, by contrast, incorporate the system's differential constraints, such as bounded turning radius, velocity, or torque. To connect two states while respecting these constraints, some planners [17] employ a steering function [5, 18]—which computes a dynamically feasible trajectory between two configurations. However, exact steering functions are only available for a limited class of systems (e.g., linear model or a Dubins car).

In contrast,tree-based kinodynamic SBPs [19, 20, 21] span a search tree by extending its branches by simulating random actions, thus ensuring the generated edges adhere to the dynamics. Modern approaches attempt to enhance efficiency by incorporating heuristics and pruning [20, 22, 21] while ensuring (asymptotic) optimality [23, 21, 24], meaning they converge to an optimal solution as the number of samples approaches infinity. While these methods allow scalable search for trajectories, their exhaustive exploration can be inefficient in large spaces or systems with high degrees of freedom.

Learning-based approaches [25] leverage past experience to improve solution quality and reduce computational effort when solving new planning problems. The seminal work [8] along with numerous follow-up studies [26, 27, 28, 29, 30], explore the use of diffusion models (DMs) [31, 32, 33] as well as the recent Flow Matching (FM) [34, 35]. DMs have demonstrated strong capabilities in planning, particularly due to their ability to sample from high-dimensional, multi-modal trajectory distributions. However, we identify several critical limitations that remain widespread:

- **Lack of Collision Avoidance Guarantees**. Trajectory prediction via DMs could result in collisions with the environment or the robot itself. Although custom controllers for collision avoidance during physical manipulator experiments can be employed [26], they are difficult to craft for agile fast-moving robots. Other methods [27, 28, 36, 30, 37, 38, 39] mitigate the issue by guiding the denoising process away from obstacles, but they do not guarantee the final output to be completely collision-free, even after post processing refinement. In contrast, the incorporation of collision checking in DiTree means our search tree is inherently collision-free.
- **Limited Out-of-Distribution (OOD) Performance**. As is endemic to many learning-based approaches, DMs tend to catastrophically fail when confronted with environments that diverge sharply from the training set. Some methods [40, 41] attempt to alleviate this issue through hierarchical planning, yet generalization is largely confined to environments resembling the training set, with the work [28] explicitly opting to specialize in a single environment due to the impractical data demands of broader generalization. In comparison, We empirically demonstrated that for DiTree training on a single maze is sufficient to generalize to wildly different environments.
- **Predefined Trajectory Horizon**. DMs operate by denoising a fixed-length input, requiring the trajectory horizon to be specified *a priori*. A common workaround is to use the DM as a Model Predictive Control (MPC) policy [26], replanning iteratively over a short horizon. Other approaches [27, 28, 42, 40] select a predefined heuristic horizon, either committing to it or interpolating between predicted states but losing their temporal significance. In contrast, DiTree's tree-based search dynamically expands edges until the goal is reached, sidestepping these limitations.
- **Lack of Differential Constraints**. Many diffusion-based approaches [8, 42, 29] directly predict trajectory states without explicitly enforcing differential constraints in the final outputs, thus requiring an external local planner to avoid violation of the robots kinodynamic constraints [42]. In contrast, our approach predicts actions and applies them directly to the robot's dynamic model, ensuring inherent adherence to system dynamics.

Hybrid approaches have also explored the integration of learning and search. While AlphaZero [11] is a notable example, its reliance on MCTS impedes its use in continuous-action KMP, and its learned value function restricts its generalization to new environments. Most other works focus on motion planning without differential constraints, or address higher-level task planning [43, 44, 45, 46, 47]. Existing methods which do address KMP typically combine a low-level learned policy with a high-level tree search, but provide the policy with either no obstacle information [48, 49] or only limited observation from onboard sensors [50]. Notably, RL-RRT [50] acknowledge reliance on simulated LIDAR measurements alone hampers planning in complex maze-like environments, especially those unseen during training (see their Fig. 7). In contrast, our diffusion model is privy to the entire local map regardless of visibility. Furthermore, to the best of our knowledge, none of these methods address planning challenges for robots characterized by large state or action spaces.

3 Preliminaries and Problem Definition

Kinodynamic Motion Planning. Consider a robot operating within a continuous state space $\mathcal{X} \subseteq \mathbb{R}^n$ and a control space $\mathcal{U} \subseteq \mathbb{R}^m$, governed by the dynamics $\dot{x} = f(x, u), x \in \mathcal{X}, u \in \mathcal{U}$, where $f : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^n$. Denote by $\mathcal{X}_{\text{free}} \subset \mathcal{X}$ the set of collision-free states. For a start state x_{start} and goal region $\mathcal{X}_{\text{goal}} \subset \mathcal{X}$, the objective is to compute a control function $\mathbf{u} : [0, T] \to \mathcal{U}$, for some $T > 0$, that induces a trajectory $\tau : [0, T] \to \mathcal{X}$ satisfying $\tau(0) = x_{\text{start}},\ \tau(T) \in \mathcal{X}_{\text{goal}},\ \dot{\tau}(t) = f(\tau(t), \mathbf{u}(t)),\ \tau(t) \in \mathcal{X}_{\text{free}},\ \forall t \in [0, T]$. I.e., the trajectory must remain collision-free, respect the dynamics, and reach the goal region. Importantly, there is generally no closed-form solution to this problem. Consequently, sampling-based algorithms are widely used to obtain feasible trajectories.

Sampling-based Planning. Tree-based SBPs [51, 19, 20] sample states and actions to generate a dynamically-feasible tree. By construction, these algorithms ensure that any returned solution adheres to both collision-avoidance and kinodynamic constraints. We focus on this family of approaches and, unless stated otherwise, use the term SBP to refer specifically to tree-based methods. We refine this notion into a general blueprint presented in Alg. 1. Rather than depicting a specific algorithm, this outline captures the fundamental logic that underpins most kinodynamic SBPs. Our framework is applicable to any method built around this core logic, e.g., [19, 21, 20, 24].

The SBP outline begins by initializing a tree with the start state x_{start} (Line 1). We then proceed to iteratively select some tree node x_{near} (Line 3), a control input (Line 4), and employ forward integration of the dynamic model to simulate the control, termed forward propagation (Line 5), resulting in a subtrajectory π from x_{near} to x_{new}. If the resulting branch is collision free it is added to the tree. For instance, in (Kinodynamic-)RRT, node selection (Line 3) is implemented by sampling a random state $x_{\text{rand}} \in \mathcal{X}$ and identifying the nearest existing node x_{near} in the tree based on a predefined distance metric, after which action selection (Line 4) is done using uniform random sampling.

Algorithm 1 Sampling-based Tree Planner

```
1: $\mathcal{T}$.init($x_{\text{start}}$)
2: for $i = 1$ to $k$ do
3:     $x_{\text{near}} \leftarrow$ Node_Selector($\mathcal{T}$)
4:     $u \leftarrow$ Action_Selector($\mathcal{U}$)
5:     $(\pi, x_{\text{new}}) \leftarrow$ Propagate($x_{\text{near}}, u$)
6:     if Collision_Free($x_{\text{near}}, x_{\text{new}}, \mathcal{X}_{\text{obs}}$) then
7:         $\mathcal{T}$.add_vertex($x_{\text{new}}$)
8:         $\mathcal{T}$.add_edge($x_{\text{near}}, x_{\text{new}}, \pi$)
9: return $\mathcal{T}$
```

Diffusion and Flow Matching Policies. Diffusion [31, 32, 33] and flow matching [34, 35] models are generative frameworks designed to efficiently sample from complex distributions by learning a mapping $f_\theta : R^d \rightarrow R^d$ that transforms a noise sample $x_0 \sim \mathcal{N}(0, I)$ to a sample from the target distribution $x_1 \sim D$. Both models excel at generating complex high-dimensional data such as images [52], protein backbones [53], and robotic trajectories [8, 26]. *Diffusion models* (DMs) learn to reverse a gradual noising process, where Gaussian noise is progressively added to a data point x_0 until it becomes pure noise x_1. By training a neural network to iteratively denoise x_1, it can reconstruct realistic samples from noise. *Flow matching models* (FMs) learn a velocity field $v_\theta(x, t)$ that describes the optimal direction of movement at each point: $\frac{dx}{dt} = v_\theta(x, t)$. By integrating this velocity field, samples are transported from an initial noise sample x_0 to a final structured data point x_1. FM achieves superior performance when using fewer integration steps [52, 54].

Both DMs and FMs can serve as robotic policies, referred to as Diffusion Policies [26], which generate sequences of robotic actions or future states $u_{1:N}$ by sampling from a conditional distribution $p(u|x, g, o)$. These models are often conditioned on the robot's current state x, the desired goal state g, and sensor observation o such as RGB images, depth maps [26], or point clouds [55].

4 Diffusion Tree Algorithm

We present our DiTree approach for leveraging diffusion models (DMs) for efficient kinodynamic motion planning (KMP) and discuss its theoretical implications.

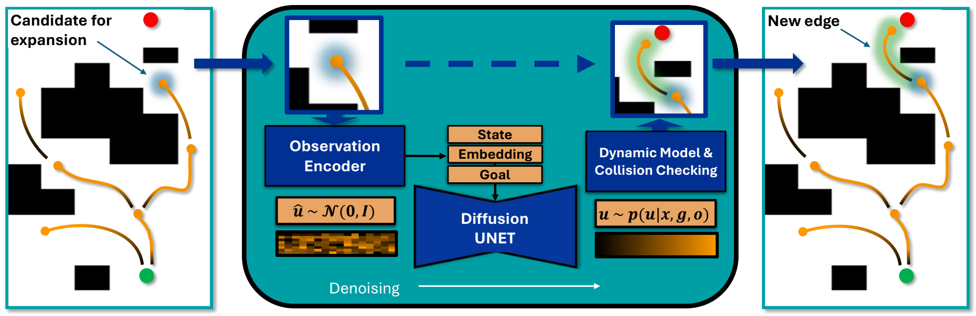


Figure 2: Action sampling in DiTree: (Left) A candidate node is selected for expansion and a local observation is extracted from its surroundings. (Center) A DM conditioned on the observation, current robot state, and goal, generates an action sequence which is simulated and checked for collision. (Right) The new edge is added to the tree. Start indicated by ●, goal by ●.

4.1 Action Sampling with Diffusion Policy

Our framework enhances methods compatible with Alg. 1, by implementing its action selection procedure (Line 4) using a context-aware sampling procedure. Instead of sampling a single action from a uniform distribution, we sample a sequence of N actions $u_{1:N}$, drawn from a distribution informed by the planning context. Inspired by recent advances in diffusion-based planning [8, 26], we implement action sampling $u_{1:N} \sim p\big(u_{1:N} \mid x_{\text{near}}, x_{\text{target}}, \mathcal{X}_{\text{obs}}^{\text{near}}\big)$ as inference from a *conditional diffusion policy* (see Fig. 2). The policy is trained on expert trajectories and conditioned on three key inputs: (1) the current tree state x_{near}, (2) a target state (which is either a goal or an exploration-guiding state) x_{target}, and (3) local obstacle information $\mathcal{X}_{\text{obs}}^{\text{near}}$ with respect to x_{near}. During training, x_{target} is set to the scenario's goal state, reinforcing goal-directed behavior. However, at test time, we introduce a *diffusion goal bias (DGB)* to promote exploration, replacing the goal state with a randomly sampled intermediate state x_{rand}. This encourages the policy to explore alternative regions of the space, generating more diverse actions and improving search coverage.

To foster generalization in our approach, we simplify the learning problem through two key strategies: (1) As effective planning should be agnostic to arbitrary global coordinates, we represent x_{target} relative to the chosen state x_{near}, introducing *translation and rotation invariance*. (2) As training a sampler that effectively conditions on all obstacles in a scene across arbitrary scenarios requires a highly diverse and extensive dataset, we condition instead the DP on a *localized, state-dependent subset* of obstacles, denoted as $\mathcal{X}_{\text{obs}}^{\text{near}}$, extracted relative to the frame of x_{near}.

After action sampling, a new edge for tree expansion is generated by executing the sequence $u_{1:N}$ via forward propagation, where each action u_i is applied for some dt. The resulting trajectory segment is accepted only if it passes collision checking, otherwise it is discarded. To ensure full information utilization during edge generation—despite relying only on local observations—we employ DM as an MPC policy during forward propagation. I.e., the policy iteratively resamples actions based on the updated local observations, enabling a flexible generation of arbitrarily long edges by perpetual sampling and forward propagation even with a very local view. The process continues until a termination condition is met, such as reaching a predefined propagation duration.

Action sampling with diffusion models is substantially more time-consuming than other components of Alg. 1, becoming the primary bottleneck in per-iteration runtime. This challenges their integration into sampling-based planners (SBPs), which require frequent sampling to drive exploration. Unlike standalone MPC-style diffusion policies [26], where the number of denoising steps is tuned to match a target control frequency, tree-based planners have no natural temporal anchor for setting the sampling schedule. Instead, the objective is to maximize planning success within a fixed time budget, requiring a careful balance between sampling speed and action quality. Fast sampling enables exploration of multiple branches, while not sacrificing the quality needed to capture the multimodal distribution of feasible edges. In Sec. 5, we empirically investigate this trade-off and demonstrate that flow matching offers an effective balance between efficiency and performance. Consequently, hyperparameters such as the number of denoising steps must be reconsidered for the unique demands of tree-based search.

4.2 Theoretical Guarantees

Some SBPs have guarantees such as *probabilistic completeness* (PC) [56] and *asymptotic optimality* (AO) [20], which rely on the *full support* of uniform random action sampling. Although DiTree uses non-uniform diffusion-based samples from a learned distributions, we demonstrate that the favorable SBP properties can be transferred to our approach, based on our proof that diffusion models posses full support. We prove the following result for RRT-style node selection (line 3).

Theorem 1. *Consider a Lipschitz-continuous system $f : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^n$, and suppose that the action sampler $u_{1:N} \sim p(u_{1:N} \mid \cdot)$ has full support. Then RRT-based DiTree is PC: there exist constants $a, b > 0$ such that for any robust KMP problem, DiTree finds a solution with probability $\geq 1 - ae^{-bk}$, where k is the number of samples.*

The full proof, which relies on an intimate understanding of RRT's behavior, along with a derivation of full support for commonly-used DMs and FMs, are provided in Appendix 7. We also discuss how asymptotic optimality is achieved when pairing DiTree with planners such as SST [20] and AO-RRT [21].

5 Experiments

To assess generalization and performance, we test DiTree against several baselines across 15 distinct scenarios on unseen maps (Fig. 3), using two robot types. Each method is run for 20 trials per scenario with a 120-second time limit. Performance is measured by the success rate (i.e., the proportion of trials that result in a collision-free path) and the average planning time over successful trials.

We implement an instance of DiTree in Python. As an SBP backbone we build upon RRT [19] due to its simplicity and popularity. For the diffusion policy we adopt single-step flow matching (FM) [34], due to a favorable trade-off between speed and quality compared to diffusion models [54]. As a single planning query could require running inference hundreds of times, we prioritize inference speed over absolute quality, but explore this trade-off in our ablation study. While more elaborate methods for single-step inference exist [57, 58], they introduce a more complex training pipeline. For our local observations we use an occupancy grid (Fig 2). Transformers are widely used and generally yield superior results, yet they are also highly sensitive to hyperparameters [26, 59]. We leave their integration for future work and instead use UNet [60].

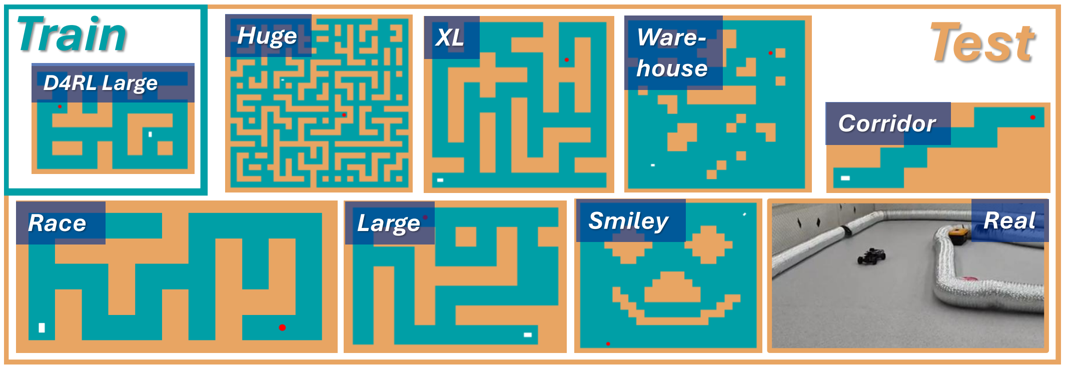


Figure 3: Experiment scenarios: We train on a single map, D4RL *AntMaze Large* (Top Left), and test on a variety of unseen maps. Queries shown for reference, car in white and goal in ●.

Robots. We evaluate our method on two robotic domains: our custom *CarMaze* (implemented using the CasADi framework [61, 62]) and D4RL's *AntMaze* (part of the D4RL benchmark; based on the MuJoCo physics engine [63, 64]), both operating within 2D maze layouts. CarMaze relies on a 6D non-holonomic ground vehicle modeled using a single-track dynamic system [62], controlled via the throttle rate and steering rate. AntMaze consists of a 29D quadruped robot with a 8D action space specifying joint torques (Fig. 1). See App. A.4 for more details. For each robot type, an FM policy is trained on an offline dataset collected from a *single* environment, D4RL's [64, 65] *Large* maze. To encourage relevant planning behaviors, we filter out states in close proximity to obstacles. Training and testing were conducted on an RTX 3090 GPU. Training lasts 2 hours.

Baselines. We compare DiTree against the two classical SBPs RRT and SST, which are implemented in OMPL [66]—a leading C++ implementation. For a learning-based baseline, as far as we know there are currently no available implementations of kinodynamic methods that could be trained on a single environment and generalize to others. Therefore, we compare against a standalone DP [26],

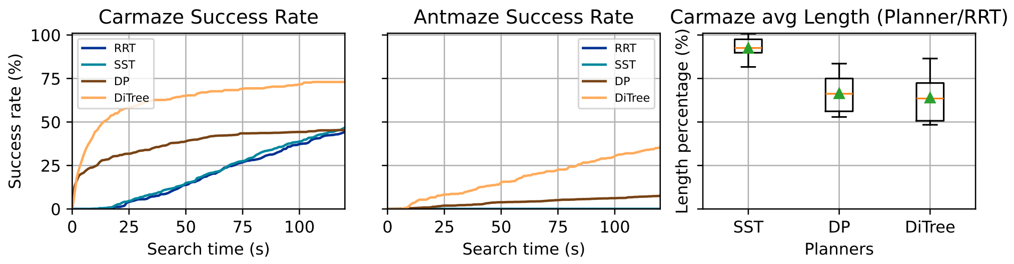


Figure 4: (Left and middle) Average success rate vs. search time across all test scenarios for CarMaze and AntMaze, comparing different methods. (Right) Comparison of the distribution of average trajectory lengths across scenarios, relative to RRT.

using the same model checkpoint as DiTree. This baseline highlights the difference between direct trajectory generation and action sampling in an SBP. For the DP setup, the policy employs the dynamic model for forward propagation of sampled action sequences, but implements search by repeatedly generating rollouts until either the goal is reached without collision or timing out.

5.1 Results

Scenario	CarMaze (SR (%) ↑ / RT (s) ↓)				AntMaze (SR (%) ↑ / RT (s) ↓)	
	RRT	SST	DP	DiTree(ours)	DP	DiTree(ours)
Race	0 / –	0 / –	5 / 93.5 ± 0.0	**45 / 40.3** ± 24.6	0 / –	**30 / 87.7** ± 18.4
Warehouse 1	50 / 90.2 ± 22.4	55 / 95.1 ± 19.6	0 / –	**95 / 21.7** ± 15.5	0 / –	**45 / 87.2** ± 17.7
Warehouse 2	20 / 91.2 ± 23.1	65 / 98.4 ± 17.4	**100** / 23.2 ± 19.8	**100 / 9.3** ± 7.5	0 / –	**25 / 94.3** ± 14.0
Warehouse 3	95 / 65.5 ± 19.0	90 / 68.3 ± 17.5	70 / 51.0 ± 26.1	**100 / 31.3** ± 31.5	0 / –	**10 / 106.3** ± 6.4
Corridor	**100** / 66.8 ± 20.0	**100** / 57.1 ± 24.4	**100** / 24.2 ± 20.1	10 / 44.2 ± 23.0	60 / 47.4 ± 39.8	**100 / 19.1** ± 19.8
Huge 1	35 / 88.2 ± 19.8	**50 / 70.5** ± 17.5	0 / –	5 / 115.2 ± 0.0	0 / –	**50 / 62.2** ± 25.5
Huge 2	50 / 58.6 ± 31.6	50 / 56.7 ± 28.5	**100 / 0.5** ± 0.4	**100** / 4.0 ± 5.7	**35** / 66.0 ± 32.2	20 / **20.3** ± 13.6
Huge 3	95 / 61.0 ± 18.3	85 / 50.5 ± 18.9	0 / –	**100 / 14.0** ± 16.0	5 / **19.6** ± 0.0	**80** / 51.6 ± 31.7
Large 1	65 / 91.2 ± 14.3	50 / 95.6 ± 16.7	**100 / 2.1** ± 1.7	**100** / 19.9 ± 27.1	10 / 74.4 ± 31.5	**50 / 66.6** ± 27.8
Large 2	0 / –	0 / –	5 / **12.9** ± 0.0	**45** / 41.7 ± 29.9	0 / –	**20 / 63.2** ± 4.7
XL 1	50 / 79.0 ± 19.4	55 / 79.9 ± 21.3	30 / **33.2** ± 19.4	**75** / 50.7 ± 29.2	0 / –	**60 / 66.2** ± 22.9
XL 2	0 / –	0 / –	0 / –	**65 / 43.0** ± 24.1	**0** / –	**0** / –
XL 3	0 / –	0 / –	0 / –	**60 / 51.0** ± 26.9	**0** / –	**0** / –
Smiley 1	15 / 95.4 ± 21.9	10 / 71.7 ± 32.1	80 / 51.4 ± 30.2	**100 / 8.6** ± 9.5	0 / –	**15 / 97.3** ± 14.2
Smiley 2	**100** / 31.0 ± 12.8	**100** / 29.6 ± 14.6	**100 / 0.7** ± 0.6	**100** / 5.1 ± 9.1	0 / –	**30 / 74.9** ± 22.7
Average	45.0 / 67.3 ± 27.5	47.3 / 66.2 ± 29.6	46.0 / 20.7 ± 26.9	**73.3** / 23.3 ± 26.9	7.3 / **54.5** ± 38.0	**35.7** / 59.4 ± 34.0

Table 1: Combined comparison for CarMaze (RRT, SST, DP, DiTree) and AntMaze (DP, DiTree) across different scenes. Each cell displays the ratio success-rate / run-time (mean ± std), best result per scenario is **highlighted**. RRT and SST failed on all AntMaze trials.

Our results (Table 1), show that DiTree achieves a strong performance across a range of out-of-distribution (OOD) scenarios, combining the speed of learned policies with the reliability of structured search. DiTree matches the runtime of DP–4× faster than classical SBPs—while outperforming both in terms of success rate (on average). On CarMaze, DiTree achieves a 26% higher success rate compared to all other methods, while in the AntMaze domain—where classical SBPs fail to produce any valid trajectories—it outperforms DP by a margin of 28%. While this work does not tackle optimal planning, a comparison of relative trajectory lengths (Fig 4) shows DiTree discovers 25-50% shorter trajectories compared to RRT, underscoring its strength in producing efficient solutions and showing promising potential for integration of asymptotically-optimal SBPs as the planning backbone. These results demonstrate the advantage of integrating learned priors with guided exploration, particularly in environments characterized by complex dynamics and constrained, narrow passages.

There are a few CarMaze scenarios where DiTree underperforms. In the *Corridor* scenario, for example, DiTree achieves only a 10% success rate compared to 100% for all other methods. Examining the search tree reveals that most of the tree nodes have reached configurations from which the car cannot possibly navigate towards the goal without collision. Upon sampling of such nodes, precious iterations are wasted in such local traps. This inefficiency could potentially be mitigated by incorporating pruning and advanced node selection strategies.

5.2 Ablation Study

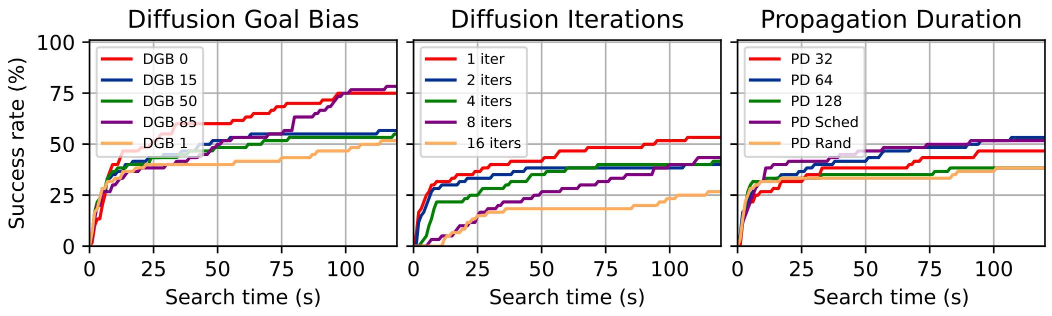


Figure 5: Comparison of Success Rate vs Runtime for different ablation tests.

We conduct an ablation study using our validation scenarios to assess the impact of key design choices specific to our framework. Results are presented in Fig 5. First, we vary the number of *denoising iterations* used to generate a sample with FM model, exploring the trade-off between sampling speed and action quality. Surprisingly, we find a *single diffusion iteration* yields the best overall performance, which we subsequently chose for our main experiments. This stands in contrast to traditional diffusion-based planners, where even SOTA FM policies [54] require several iterations to produce high-quality actions necessary for successful rollouts. In DiTree, however, faster, coarser samples are preferable, as they provide timely guidance for tree expansion without incurring the computational cost of multiple denoising steps. These results highlight a key distinction in how generative models are used within our framework: not to execute entire trajectories, but to prioritize promising search directions.

Next, we ablate the effect of a key hyperparameter of DiTree, *propagation duration* (N). The propagation duration is defined as the length of the action sequence forming a tree edge. A shorter duration limits per-edge advancement—quickly shifting focus between nodes to broaden overall exploration—whereas a longer duration can cover more ground but risk inefficiency if a collision occurs. We compare several strategies for setting N: (i) a random selection between 32–128 steps per iteration, (ii) fixed values of 32, 64, or 128 steps, and (iii) a scheduled approach that gradually shortens edge length $(128 \rightarrow 96 \rightarrow 64)$ upon repeated node visits. Our experiments show a fixed value of 64 gave the best average results across our scenarios by a slight margin.

Finally, we examine the role of *diffusion goal bias* (DGB)[2]—a hyperparameter controlling the ratio of conditioning the FM on x_{rand} versus the final goal x_{goal}. We find that setting DGB to either 0% or 85% yields the highest overall success rates. We observe complex scenarios benefit from a DGB of 0% due to the extensive exploration, however, simpler scenarios fall short as excessive random exploration leads to markedly longer and less efficient trajectories. Thus, in our experiments we opt for 85% promoting rapid expansion toward the goal while maintaining sufficient exploration.

5.3 Real-World Applicability

We demonstrate integration with a physical robot by comparing DiTree to RRT in a sharp-corner turning car scenario (Fig 3). For deployment, output trajectories are smoothed and tracked using a pure-pursuit controller, without further optimization. Each planner is tested over ten runs (Appendix A.4). During execution, RRT encounters eight collisions, while DiTree consistently steers clear of obstacles. As the underlying algorithm is identical for both methods, this highlights the robustness and quality of sampling learned from the expert dataset, whose trajectories inherently maintain a safety margin—mirroring the strong performance of diffusion-based approaches in driving tasks demonstrated by prior work [67].

6 Conclusion

We introduced DiTree—a novel framework that integrates diffusion models with sampling-based tree search to address the challenges of KMP. By leveraging learned, environment-aware priors for action sampling, DiTree balances the strengths of generative models and classical planners—achieving fast, generalizable, and collision-free planning, while preserving guarantees of the underlying planner.

We aim for this work to lay the groundwork for safer, more general, data-driven motion planners in robotics. Future work includes inference optimization using distillation and quantization [68]. Other potential improvements involve learning other SBP components like node selection, as well as better leveraging GPU parallelization during search. As our work diverts the computational bottleneck of SBPs from collision detection [69] to action sampling via DM (Appendix D.1.2), it motivates the development of new tailored models that exploit the problem structure to facilitate faster solutions of multiple queries.

[2]Not to be confused with RRT's node selection goal bias.

7 Limitations

Although DiTree demonstrates strong performance, several assumptions limit its ability to reach its full potential in more complex settings.

First, we assume full knowledge of all obstacles at planning time, including their geometry. This restricts applicability in dynamic or partially observed settings. One remedy is to extend Alg. 1 to support partial knowledge of the environment [70, 71] and replan over short horizons using updated observations.

Second, in our implementation, we rely on an approximation of the dynamic model, which could lead to reduced performance in practice, especially for complex real-world systems with chaotic dynamics or modeling uncertainty. Those issues could mitigated by learning a more accurate representation of the model [72], or designing a learned controller explicitly reasoning about dynamic residual between the simulation and the real world [73].

Third, our workspace is strictly two-dimensional, as is represented via occupancy grids. While sufficient for planar navigation tasks, it does not capture the full geometry of 3D environments; incorporating richer modalities such as depth images or point clouds [55] could address this limitation.

Fourth, our diffusion model is trained on pre-collected expert demonstrations. This enables straightforward training but limits applicability in environments lacking demonstration data. Extending our approach to learn behaviors from scratch using reinforcement learning [74] is a promising direction.

Lastly, we employ a bare-bone implementation of sampling-based planners that lack optimization and fine-grained parallelism, limiting our ability to fully exploit batch inference in diffusion models. Incorporating recent parallelized frameworks such as [75, 76] could significantly improve runtime efficiency.

Acknowledgments

We would like to thank Ori Menashe for his valuable assistance with the experiments, and Tomer Michaeli for his insightful comments on theoretical aspects of diffusion models.

References

[1] B. Paden, M. Cáp, S. Z. Yong, D. S. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1:33–55, 2016. URL `https://api.semanticscholar.org/CorpusID:1229096`.

[2] S. Ghambari, M. Golabi, L. Vermeulen-Jourdan, J. Lepagnot, and L. Idoumghar. UAV path planning techniques: a survey. *RAIRO Oper. Res.*, 58:2951–2989, 2024. URL `https://api.semanticscholar.org/CorpusID:268718404`.

[3] C. R. de Lima, S. G. Khan, M. Tufail, S. H. Shah, and M. R. O. A. Maximo. Humanoid robot motion planning approaches: a survey. *Journal of Intelligent & Robotic Systems*, 110(2):86, 2024. URL `https://doi.org/10.1007/s10846-024-02117-z`.

[4] R. "Tedrake. *"Underactuated Robotics"*. 2023. URL `"https://underactuated.csail.mit.edu"`.

[5] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, UK, 2006.

[6] R. S. Sutton. The bitter lesson. urlhttp://www.incompleteideas.net/IncIdeas/BitterLesson.html, 2019.

[7] M. Elbanhawi and M. Simic. Sampling-based robot motion planning: A review. *Ieee access*, 2:56–77, 2014.

[8] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022.

[9] S. Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review, 2018. URL https://arxiv.org/abs/1805.00909.

[10] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017. URL https://www.nature.com/articles/nature24270.

[11] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. URL https://www.science.org/doi/abs/10.1126/science.aar6404.

[12] OpenAI. Openai o1 system card, 2024. URL https://openai.com/index/openai-o1-system-card/.

[13] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

[14] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[15] R. Bohlin and L. Kavraki. Path planning using lazy prm. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, pages 521–528 vol.1, 2000.

[16] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30:846 – 894, 2011. URL https://api.semanticscholar.org/CorpusID:14876957.

[17] D. J. Webb and J. van den Berg. Kinodynamic rrt*: Optimal motion planning for systems with linear differential constraints. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[18] H. B. Keller. *Numerical methods for two-point boundary-value problems*. Courier Dover Publications, 2018.

[19] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *I. J. Robotics Res.*, 20 (5):378–400, 2001.

[20] Y. Li, Z. Littlefield, and K. E. Bekris. Asymptotically optimal sampling-based kinodynamic planning. *Int. J. Rob. Res.*, 35(5):528–564, Apr. 2016. URL https://doi.org/10.1177/0278364915614386.

[21] K. Hauser and Y. Zhou. Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space. *IEEE Transactions on Robotics*, 32(6):1431–1443, 2016.

[22] I. A. Şucan and L. E. Kavraki. *Kinodynamic Motion Planning by Interior-Exterior Cell Exploration*, pages 449–464. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. URL https://doi.org/10.1007/978-3-642-00312-7_28.

[23] M. Kleinbort, E. Granados, K. Solovey, R. Bonalli, K. E. Bekris, and D. Halperin. Refined analysis of asymptotically-optimal kinodynamic planning in the state-cost space. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6344–6350, 2019. URL https://api.semanticscholar.org/CorpusID:212634333.

[24] M. Fu, K. Solovey, O. Salzman, and R. Alterovitz. Resolution-optimal motion planning for steerable needles. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 9652–9659, 2022.

[25] G. P. d. Carvalho, T. Sawanobori, and T. Horii. Data-driven motion planning: A survey on deep neural networks, reinforcement learning, and large language model approaches. *IEEE Access*, 13:52195–52245, 2025.

[26] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 2024.

[27] K. Saha, V. Mandadi, J. Reddy, A. Srikanth, A. Agarwal, B. Sen, A. Singh, and M. Krishna. Edmp: Ensemble-of-costs-guided diffusion for motion planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, page 10351–10358. IEEE, May 2024. URL `http://dx.doi.org/10.1109/ICRA57147.2024.10610519`.

[28] J. Carvalho, A. Le, M. Baierl, D. Koert, and J. Peters. Motion planning diffusion: Learning and planning of robot motions with diffusion models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[29] W. Xiao, T.-H. Wang, C. Gan, R. Hasani, M. Lechner, and D. Rus. Safediffuser: Safe planning with diffusion probabilistic models. In *International Conference on Learning Representations*, 2025.

[30] Y. Luo, C. Sun, J. B. Tenenbaum, and Y. Du. Potential based diffusion motion planning. In *International Conference on Machine Learning*, pages 33486–33510. PMLR, 2024.

[31] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. pmlr, 2015.

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

[33] Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[34] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling, 2023. URL `https://arxiv.org/abs/2210.02747`.

[35] X. Liu, C. Gong, and Q. Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

[36] M. Seo, Y. Cho, Y. Sung, P. Stone, Y. Zhu, and B. Kim. Presto: Fast motion planning using diffusion models based on key-configuration environment representation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[37] K. Lee, S. Kim, and J. Choi. Refining diffusion planner for reliable behavior synthesis by automatic detection of infeasible plans. In *Advances in Neural Information Processing Systems*, 2023.

[38] S. Huang, Z. Wang, P. Li, B. Jia, T. Liu, Y. Zhu, W. Liang, and S.-C. Zhu. Diffusion-based generation, optimization, and planning in 3d scenes. *arXiv preprint arXiv:2301.06015*, 2023.

[39] L. Feng, P. Gu, B. An, and G. Pan. Resisting stochastic risks in diffusion planners with the trajectory aggregation tree. In *International Conference on Machine Learning*, volume 235, pages 13175–13198. PMLR, 2024.

[40] C. Chen, F. Deng, K. Kawaguchi, C. Gulcehre, and S. Ahn. Simple hierarchical planning with diffusion. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=kXHEBK9uAY.

[41] Z. Dong, J. Hao, Y. Yuan, F. Ni, Y. Wang, P. Li, and Y. Zheng. Diffuserlite: Towards real-time diffusion planning, 2024. URL https://arxiv.org/abs/2401.15443.

[42] J. Liu, M. Stamatopoulou, and D. Kanoulas. Dipper: Diffusion-based 2d path planner applied on legged robots. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9264–9270. IEEE, 2024.

[43] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip. Motion planning networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2118–2124. IEEE, 2019.

[44] B. Ichter, J. Harrison, and M. Pavone. Learning sampling distributions for robot motion planning. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094, 2017.

[45] J. Wang, W. Chi, C. Li, C. Wang, and M. Q. Meng. Neural rrt*: Learning-based optimal path planning. *IEEE Transactions on Automation Science and Engineering*, 17:1748–1758, 2020. URL https://api.semanticscholar.org/CorpusID:215841350.

[46] B. Chen, B. Dai, Q. Lin, G. Ye, H. Liu, and L. Song. Learning to plan in high dimensions via neural exploration-exploitation trees. In *International Conference on Learning Representations*, 2019. URL https://api.semanticscholar.org/CorpusID:214300994.

[47] B. Ichter, P. Sermanet, and C. Lynch. Broadly-exploring, local-policy trees for long-horizon task planning. In *Conference on Robot Learning*, 2020. URL https://api.semanticscholar.org/CorpusID:222310776.

[48] A. Sivaramakrishnan, E. Granados, S. Karten, T. McMahon, and K. E. Bekris. Improving kinodynamic planners for vehicular navigation with learned goal-reaching controllers. *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9038–9043, 2021. URL https://api.semanticscholar.org/CorpusID:238531708.

[49] T. McMahon, A. Sivaramakrishnan, K. Kedia, E. Granados, and K. Bekris. Terrain-aware learned controllers for sampling-based kinodynamic planning over physically simulated terrains. pages 2925–2930, 10 2022.

[50] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust. Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies. *IEEE Robotics and Automation Letters*, 4:4298–4305, 2019. URL https://api.semanticscholar.org/CorpusID:195874334.

[51] J. J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[52] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*, 2024.

[53] A. J. Bose, T. Akhound-Sadegh, G. Huguet, K. Fatras, J. Rector-Brooks, C.-H. Liu, A. C. Nica, M. Korablyov, M. Bronstein, and A. Tong. Se (3)-stochastic flow matching for protein backbone generation. *arXiv preprint arXiv:2310.02391*, 2023.

[54] X. Hu, B. Liu, X. Liu, and Q. Liu. Adaflow: Imitation learning with variance-adaptive flow-based policies. *ArXiv*, abs/2402.04292, 2024. URL https://api.semanticscholar.org/CorpusID:267523297.

[55] Y. Ze, G. Zhang, K. Zhang, C. Hu, M. Wang, and H. Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

[56] M. Kleinbort, K. Solovey, Z. Littlefield, K. E. Bekris, and D. Halperin. Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation. *IEEE Robotics and Automation Letters*, 4:277–283, 2018. URL https://api.semanticscholar.org/CorpusID:52303415.

[57] L. Yang, Z. Zhang, Z. Zhang, X. Liu, M. Xu, W. Zhang, C. Meng, S. Ermon, and B. Cui. Consistency flow matching: Defining straight flows with velocity consistency. *arXiv preprint arXiv:2407.02398*, 2024.

[58] K. Frans, D. Hafner, S. Levine, and P. Abbeel. One step diffusion via shortcut models. *ArXiv*, abs/2410.12557, 2024. URL https://api.semanticscholar.org/CorpusID:273375140.

[59] S. Dasari, O. Mees, S. Zhao, M. K. Srirama, and S. Levine. The ingredients for robotic diffusion transformers. *ArXiv*, abs/2410.10088, 2024. URL https://api.semanticscholar.org/CorpusID:273346315.

[60] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24574-4.

[61] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. CasADi - A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, (11):1–36, Mar. 2019.

[62] D. Kloeser, T. Schoels, T. Sartor, A. Zanelli, G. Prison, and M. Diehl. Nmpc for racing using a singularity-free path-parametric model with obstacle avoidance. *IFAC-PapersOnLine*, 53(2): 14324–14329, 2020. URL https://www.sciencedirect.com/science/article/pii/S2405896320317845.

[63] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[64] R. de Lazcano, K. Andreas, J. J. Tai, S. R. Lee, and J. Terry. Gymnasium robotics, 2024. URL http://github.com/Farama-Foundation/Gymnasium-Robotics.

[65] Minari Contributors. Minari: A dataset API for Offline Reinforcement Learning, May 2023.

[66] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. https://ompl.kavrakilab.org.

[67] Y. Zheng, R. Liang, K. ZHENG, J. Zheng, L. Mao, J. Li, W. Gu, R. Ai, S. E. Li, X. Zhan, and J. Liu. Diffusion-based planning for autonomous driving with flexible guidance. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=wM2sfVgMDH.

[68] T. Liang, C. J. Glossner, L. Wang, and S. Shi. Pruning and quantization for deep neural network acceleration: A survey. *ArXiv*, abs/2101.09671, 2021. URL https://api.semanticscholar.org/CorpusID:231699188.

[69] M. Kleinbort, O. Salzman, and D. Halperin. Collision detection or nearest-neighbor search? On the computational bottleneck in sampling-based motion planning. In *Workshop on the Algorithmic Foundations of Robotics*, pages 624–639. Springer, 2016.

[70] A. Elhafsi, B. Ivanovic, L. Janson, and M. Pavone. Map-predictive motion planning in unknown environments. In *IEEE International Conference on Robotics and Automation*, pages 8552–8558. IEEE, 2020.

[71] S. Koenig and M. Likhachev. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3):354–363, 2005.

[72] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 36(4):1307–1319, 2020.

[73] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

[74] M. Ghasemi, A. H. Moosavi, and D. Ebrahimi. A comprehensive survey of reinforcement learning: From algorithms to practical challenges, 2025. URL https://arxiv.org/abs/2411.18892.

[75] C. H. Huang, P. Jadhav, B. Plancher, and Z. Kingston. prrtc: Gpu-parallel rrt-connect for fast, consistent, and low-cost motion planning, 2025. URL https://arxiv.org/abs/2503.06757.

[76] W. Thomason, Z. Kingston, and L. E. Kavraki. Motions in microseconds via vectorized sampling-based planning. In *IEEE International Conference on Robotics and Automation*, pages 8749–8756. URL http://arxiv.org/abs/2309.14545.

[77] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[78] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=PxTIG12RRHS.

[79] V. I. Arnold. *Ordinary Differential Equations*. Springer, 3rd edition, 1992. ISBN 9783540548133.

[80] O. Nechushtan, B. Raveh, and D. Halperin. Sampling-diagram automata: A tool for analyzing path quality in tree planners. In *International Workshop on the Algorithmic Foundations of Robotics*, volume 68 of *Springer Tracts in Advanced Robotics*, pages 285–301. Springer, 2010.

[81] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

[82] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. URL https://doi.org/10.1038/s41592-019-0686-2.

Appendix A: Theoretical Guarantees

In this section, we prove Theorem 1, namely that DiTree, when using an RRT-based node selection mechanism, is PC, under the assumption that the underlying action sampling mechanism has full support. To obtain the proof, we review the main ingredient in the PC proof of the classic RRT algorithm and identify necessary changes to our setting. We then demonstrate that diffusion models satisfy the full support assumption. We conclude the section with a discussion about obtaining AO guarantees when using AO-RRT or SST as underlying SBPs.

A.1 Probabilistic Completeness of RRT

We review the main ingredients in RRT's PC proof [56, Theorem 2]. The proof assumes that (i) the dynamics f are Lipschitz continuous with respect to states and controls [56, Definition 2], and (ii) that there is a *robust* solution to the KMP problem, i.e., a solution trajectory with positive clearance. (Correspondingly, we make those common assumptions in our Theorem 1 as well.)

The proof then proceeds in the following manner. Denote by π the robust nominal solution trajectory, T its duration, and $\mathbf{u}$ its control function. A set of landmark states $x_0, \ldots, x_m$ is chosen along π, where $x_i := \pi(i \cdot \tau)$ and τ is a constant duration, where $x_0 = x_{\text{start}}$ and $x_m \in \mathcal{X}_{\text{goal}}$. Then, each landmark is associated with a radius r_i-ball centered at x_i, denoted by $B_{r_i}(x_i) \subset \mathcal{X}$.

It is shown that RRT visits each of those balls, with probability at least $1 - a'e^{-b'k}$, and thus finds a feasible solution. To achieve this, the proof computes a lower bound on the probability of making progress between two consecutive balls. Specifically, fix $0 < i < m$ and suppose that the RRT tree has vertex $x'_i \in B_{r_i}(x_i)$. Then, the probability that RRT obtains a new vertex $x'_{i+1} \in B_{\kappa r_{i+1}}(x_{i+1})$ in the current iteration, is at least $\rho_i := p_i^x \cdot p_i^t \cdot p_i^u$, for some $\kappa \in (0, 1)$.

Specifically, the value p_i^x denotes the probability of selecting an RRT node for expansion from $B_{r_i}(x_i)$ (line 4 in Alg. 1), which is at least the probability of drawing $x_{\text{rand}} \in B_{r_i 2/5}(x_i)$ uniformly at random from $\mathcal{X}$ [56, Lemma 4]. Assuming that this event occurs, i.e., $x_{\text{near}} \in B_{r_i}(x_i)$, the values p_i^t and p_i^u jointly lower-bound the probability of sampling a time duration $t_{\text{rand}} \in [0, T_{\text{prop}}]$ and action $u_{\text{rand}} \in \mathcal{U}$ such that their propagation results in a new vertex $x_{\text{new}} \in B_{\kappa r_{i+1}}(x_{i+1})$. In particular, p_i^t denotes the probability that $t_{\text{rand}} \in [T_-, T_+]$, for some constants $0 < T_- \leq T_+ \leq T_{\text{prop}}$. Similarly, p_i^u denotes the probability that u_{rand} is sampled *uniformly at random* from $B_{\Delta u}(u_i) \subset \mathcal{U}$, which is a ball in the control space of radius $\Delta u > 0$ centered at u_i, which is the nominal control at x_i [56, Lemma 3]. Importantly, the sampled action u_{rand} is applied through the entire duration t_{rand} of propagation.

The constants $\tau, m, \kappa, r_i, T_-, T_+$, and Δu depend on the nominal trajectory clearance, its duration, and the Lipschitz constants of f, but they do not depend on the number of RRT iterations k. They are chosen to ensure that the propagation described in the previous paragraph results in a collision free trajectory reaching the next ball. Moreover, those values are carefully selected such that each of the values p_i^x, p_i^t and p_i^u is greater than zero.

As a result, the probability $\rho_i > 0$ holds for any $0 \leq i \leq m$. Moreover, as it is independent of k, a Markov argument allows to lower-bound the probability that the final ball is eventually reached with $1 - a'e^{b'k}$, which concludes the proof of [56, Theorem 2].

A.2 Probabilistic Completeness of DiTree

For our proof of Theorem 1, it suffices to slightly revise the above reasoning for RRT. In particular, we keep the same construction, but alter the probability of a successful extension from ball i to ball $i + 1$, denoted by $\bar{\rho}_i := \bar{p}_i^x \cdot \bar{p}_i^t \cdot \bar{p}_i^u$, where the three values correspond in meaning to p_i^x, p_i^t and p_i^u, which we discussed above.

First, we note that $\bar{p}_i^x = p_i^x$, which follows from the fact that an RRT-based instantiation of DiTree uses the same node selection mechanism as RRT. Second, the choice of t_{rand} remains identical to RRT, therefore $\bar{p}_i^t = p_i^t$.

Thus, it remains to show that $\bar{p}_i^u > 0$. Recall that rather than sampling a single control and applying for the entire duration of the propagation, as in RRT, DiTree samples a sequence of $N > 0$ controls $u_{\text{rand}_{1:N}} \sim p_\theta\big(u_{1:N} \mid x_{\text{near}}, x_{\text{target}}, \mathcal{X}_{\text{obs}}^{\text{near}}\big)$ and propagates them sequentially, such that each remains fixed for some $\Delta t > 0$, and $N\Delta t = t_{\text{rand}}$. To mimic the behavior of RRT, we require that for any $1 \leq j \leq N$ it holds that $u_{\text{rand}_j} \in B_{\Delta u}(u_i)$ (adapting Lemma 2 in [51] to this setting is straightforward). That is, $\bar{p}_i^u := \prod_{j=1}^{N} \Pr\left(u_j \in B_{\Delta u}(u_i)\right)$, which is greater than zero if $p_\theta\big(u_{1:N} \mid x_{\text{near}}, x_{\text{target}}, \mathcal{X}_{\text{obs}}^{\text{near}}\big)$ has full support. This concludes the proof of Theorem 1.

A.3 Full Support of Diffusion and Flow Matching

Next, we justify our full support assumption, i.e., given that $u_{\text{rand}_{1:N}} \sim p_\theta\big(u_{\text{rand}_{1:N}} \mid x_{\text{near}}, x_{\text{target}}, \mathcal{X}_{\text{obs}}^{\text{near}}\big)$, where the distribution is according to a diffusion model (DMs) [31, 77] or flow matching models (FMs) [34] with weights θ, then $u_{\text{rand}_j} \in B_r(\cdot)$ for all $1 \leq j \leq N$, where the ball is of radius $r > 0$ centered around some action in $\mathcal{U}$. For simplicity, we denote the distribution from which we sample as $p_\theta(u)$, where $u := u_{\text{rand}_{1:N}}$. We also assume that the distribution is unconditioned (the derivation for conditional distributions such as DiTree's $p_\theta\big(u_{\text{rand}_{1:N}} \mid x_{\text{near}}, x_{\text{target}}, \mathcal{X}_{\text{obs}}^{\text{near}}\big)$ is straightforward and follows directly from prior work [78, 77, 34], without affecting the core argument). Without loss of generality, we consider the setting of $N = 1$, as the arguments below can be easily generalized by considering a diffusion model whose domain is $\mathbb{R}^{ND}$, rather than $\mathbb{R}^D$, which we consider below.

We provide necessary mathematical details on DMs and FMs and their sampling process, and then discuss their full support property. Specifically, we focus on the inner workings of generative sampling by integration[3] of an SDE or ODE. Both DM and FM aim to learn a transformation $T_\theta : \mathbb{R}^D \to \mathbb{R}^D$ that maps a sample u_0 from a distribution $\mathcal{N}(0, I)$[4] to a sample $u_1 \sim p_{\text{data}}$ in a complex target distribution, which in DiTree is the conditional distribution of expert action sequences. This transformation is defined over a time interval $t \in [0, 1]$ by integrating a continuous, time-dependent vector field $\mathbf{v}_\theta(u_t, t)$, parameterized by θ, that governs the evolution of the state $u_t \in \mathbb{R}^D$. The boundary conditions for this evolution are

$$u(0) = u_0, \quad u(1) = u_1 = T_\theta(u_0).$$

The nature of the vector field $\mathbf{v}_\theta(u_t, t)$ differs between the two paradigms. In DMs, $\mathbf{v}_\theta(u_t, t)$ entails an approximation of the score function, i.e., the gradient of the log-density of the time-dependent marginal $p_t(u)$:

$$\mathbf{v}_\theta(u_t, t) \propto \nabla_u \log p_t(u_t).$$

In FMs, $\mathbf{v}_\theta(u_t, t)$ is a velocity field that directly defines the transport direction and magnitude needed to continuously morph the source distribution into the target distribution across time.

In both cases, the overall transformation $T_\theta(u_0) = u_1$ is realized by solving the differential equation

$$\frac{\mathrm{d}u(t)}{\mathrm{d}t} = \mathbf{v}_\theta(u_t, t),$$

or its stochastic counterpart, depending on the formulation. The choice between a deterministic ordinary differential equation (ODE) [77] or a stochastic differential equation (SDE) [78] determines whether the generative process is deterministic or includes sampling noise.

We now detail the full support property of sampling under both the ODE and SDE formulations. Our explanation relies on the sampling dynamics themselves using a numerical solver and a learned vector field, rather than whether the specific training method used is DM or FM. Therefore, the justification below applies to both types of models, and we use them interchangeably.

[3]We emphasize this integration is *not* related to the integration of the dynamic model f used in DiTree's forward propagation (Line 5 Alg. 1).

[4]While FM could use other source distributions, in practice the normal distribution is commonly chosen.

A.3.1 ODE Formulation

In the deterministic case, the generative process is described by the associated probability flow ODE (e.g., as in [77]). Sampling $p_\theta(u)$ is performed by integrating from an initial Gaussian sample using a numerical solver. The time interval $[0, 1]$ is discretized as $t_0 < t_1 < \cdots < t_\ell$ with a time step Δt and the Euler method is applied as a numerical solver for integration over $\mathbf{v}_\theta(u_t, t)$. The value $u_{t_{i+1}}$ is derived from the previous value by the update

$$u_{t_{i+1}} = u_{t_i} + \mathbf{v}_\theta(u_{t_i}, t_i)\Delta t. \tag{1}$$

Next, we show that $u_1 = u_{t_\ell}$ has full support on $\mathbb{R}^D$. Let $u_0 \sim \mathcal{N}(0, I)$ denote the initial sample drawn from the source Gaussian, and let $T_\theta : \mathbb{R}^D \to \mathbb{R}^D$ be the transformation defined by integrating a vector field $\mathbf{v}_\theta(u_t, t)$ over time $t \in [0, 1]$, and $u_1 = T_\theta(u_0)$ the generated output. Since the vector field $\mathbf{v}_\theta(u, t)$ is globally Lipschitz and continuously differentiable, it yields a unique C^1 flow map T_θ via the Picard–Lindelöf theorem [79], which is a diffeomorphism onto its image.

To conclude, since the Gaussian distribution assigns positive probability to every non-empty open subset of $\mathbb{R}^D$, for any non-empty open set $B_r(\cdot) \subset \mathbb{R}^D$, the preimage $T_\theta^{-1}(B_r(\cdot))$ is also open and has positive measure under u_0.

A.3.2 SDE Formulation

The SDE formulation introduces sampling noise to the ODE (see, e.g., [78]), where the Euler–Maruyama method is applied to numerically simulate the SDE, which yields the update rule

$$u_{t_{i+1}} = \underbrace{u_{t_i} + v_\theta(u_{t_i}, t_i)\Delta t}_{\text{deterministic shift}} + \underbrace{g(t_i)\sqrt{|\Delta t|}\,\epsilon_i}_{\text{Gaussian noise}},$$

where $g(t)$ is the diffusion coefficient—a scalar-valued function controlling the noise magnitude at time t. It must satisfy $g(t) > 0$ for $t > 0$ to ensure well-defined noise injection. The noise term $\epsilon_i \sim \mathcal{N}(0, I)$ is a Gaussian random variable with zero mean and full-rank (positive-definite) covariance matrix, simulating the increment of a standard Wiener process (Brownian motion) over the time interval Δt.

The final update step in the sampling process $u_\ell = u_{\ell-1} + v_\theta(u_{\ell-1}, t_{\ell-1})\Delta t + g(t_{\ell-1})\sqrt{|\Delta t|}\,\epsilon_{\ell-1}$ is a summation of a deterministic term and a Gaussian noise term, which results in a Gaussian random variable with a shifted mean and non-degenerate covariance. Similarly to the ODE setting, this implies the full support of the distribution.

Remark. In other generative methods where the full support assumption might not strictly hold—for instance, if the dynamics become partially deterministic, or if the noise coefficient $g(t)$ vanishes at some positive times—full support can still be recovered by adding a small amount of independent noise to the final sample. Specifically, perturbing the output by a small Gaussian variable ensures that the resulting distribution has a strictly positive density everywhere.

A.4 Extension to Asymptotic Optimality

Our theoretical guarantees of DiTree can be further strengthened by substituting the RRT backbone planner with a planner whose solution quality is guaranteed to converge to the optimum—a property called asymptotic optimality (AO).

While RRT is PC under the assumptions of Lipschitz dynamics and the existence of a robust trajectory, it does not offer any guarantees about the *quality* of the returned path. In particular, RRT may return highly suboptimal solutions even with an infinite number of samples [80]. To address this limitation, the AO-RRT [21] was proposed. The central idea in AO-RRT is to augment the original state space $\mathcal{X}$ with an additional cost dimension. That is, AO-RRT operates in the $(d + 1)$-dimensional space $\mathcal{Y} := \mathcal{X} \times \mathbb{R}_{\geq 0}$, where each point $y = (x, c)$ stores both the state x and the cost-to-come c from x_{init} to x. The system dynamics are lifted to this space via the augmented dynamics

$$\dot{y} = F(y, u) = (f(x, u),\, g(x, u)),$$

where $g(x,u)$ is the instantaneous cost function and is assumed to be Lipschitz continuous, like f. With this formulation, AO-RRT can be viewed as RRT operating over the dynamics F in the space $\mathcal{Y}$. Thus, the AO proof of AO-RRT [23] follows the same reasoning as the PC proof of RRT [56]. Thus, by relying on the backbone of AO-RRT in DiTree, the AO property will also extend to DiTree, due to the full support of DM and FM.

Another planner that could be used in conjunction with DiTree to achieve AO is SST [20]. Although the structure of the AO proof of SST differs from that for AO-RRT, both algorithms rely on very similar building blocks. However, some care should be taken when revisiting all the steps of SST's proof, which we leave for future work.

Appendix B: Robot Dynamics

We provide additional details on the robot dynamics. The car dynamics are based on a single-track dynamic model used in [62]. This well-established model offers a good balance between physical realism and computational tractability. Compared to simplified models such as unicycle, Ackermann steering, or even kinematic bicycle models, the single-track model captures key inertial and actuation constraints. Specifically, it captures essential second-order effects such as acceleration saturation and steering inertia, which are critical for planning under dynamic constraints.

The state space is defined as $\{x, y, \psi, v, D, \delta\}$, where $\{x, y\}$ denote position, ψ the heading angle and δ the front wheel steering angle. The action space is $\{\dot{D}, \dot{\delta}\}$, representing throttle and steering rate commands. System dynamics are given by:

$$\begin{aligned}
\dot{x} &= v\cos(\psi + c_1\delta), \\
\dot{y} &= v\sin(\psi + c_1\delta), \\
\dot{\psi} &= vc_2\delta, \\
\dot{v} &= \frac{F_x}{m}\cos(c_1\delta),
\end{aligned}$$

where the longitudinal force F_x is modeled as:

$$F_x = (c_{m1} - c_{m2}v)D - c_{r2}v^2 - c_{r0}\tanh(c_{r3}v).$$

Here, m denotes the vehicle mass, c_{m1} is the motor gain, and c_{m2} captures velocity-dependent damping. The terms c_{r0} and c_{r2} model static and quadratic rolling resistance, respectively.

Mujoco's [63] ant robot is a quadruped robot consisting of four legs, each composed of two segments connected via hinge joints (Fig. 6). The upper segment of each leg connects to the torso with an additional hinge joint. The state space is 29-dimensional, comprising all joint angles, as well as the position and velocity of the ant's torso. The action space is 8-dimensional, representing the torques applied to each joint.

Appendix C: Additional Implementation Details

This section provides additional information regarding DiTree's implementation and experiments.

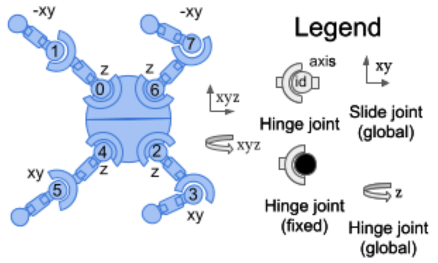


Figure 6: A schematic of the ant robot, borrowed from [81].

Data Collection. We train each robot on a dataset of expert trajectories. While the ant robot has an available dataset in D4RL [64, 65], CarMaze is not part of the D4RL suite. Thus, we generate a dataset by first planning paths using A* on a discretized grid and then executing them using a PD controller to compute dynamically feasible trajectories. Collision-free trajectories were gathered into a dataset.

Validation. Evaluating the diffusion model's generalization during training was performed by executing rollouts on a separate validation set (distinct from the test set) and analyzing performance based on the number of collisions, distance to the goal after T timesteps, and an overall qualitative assessment of path quality and diversity.

Nearest Neighbor (NN). As part of the node selection step (Line 3 of Alg. 1), RRT uses NN queries. In DiTree's RRT backbone implementation, NN queries are performed using SciPy's [82] KDTree. For our AntMaze tasks only, we chose our distance metric to exclusively account for (x, y) distance to encourage exploration and not be overshadowed by the other 27 dimensions.

Collision Checking (CC) is not explicitly available in D4RL environments. To enable CC for all the compared methods in our experiments we implement a simplified CC routine by modeling robots as sets of spheres and checking for intersections with neighboring occupancy grid cells. For experiments with OMPL, we use the available Python bindings and set state validation to use the above CC routine. We also configure OMPL to use the environments' `step()` function for forward propagation, as used by all other methods.

Appendix D: Full Experimental Results

D.1 Real-World Experiment

To evaluate our method in a real-world setting, we constructed a test environment simulating a common racing scenario: turning a corner on a track. A corresponding map was designed to reflect this geometry.

We generated 10 trajectories using both DiTree and a baseline RRT planner, omitting DP and SST since their behaviors are qualitatively similar to DiTree and RRT, respectively. During planning, we used the dynamic car model described in [62] without performing system identification. That is, we adopted the default parameters provided by the implementation. While this choice makes the resulting trajectories harder to track due to model mismatch, it more accurately reflects real-world deployment conditions, where accurate system identification is often infeasible—especially as vehicle dynamics and track conditions vary over time. All other planning and environment parameters were kept unchanged compared to the simulated experiments.

The resulting trajectories were smoothed using a cubic spline interpolation to produce smooth reference paths suitable for real-world execution. A standard pure-pursuit controller was used for trajectory tracking on the physical platform. Real-time state estimation was provided by an OptiTrack motion capture system.

Our experiments show classical SBPs trajectories are more difficult to track, which often resulted in collisions. During execution of RRT trajectories, 8 trials out of 10 experienced at least one collision with an obstacle. Collisions were either due to controller overshoot near tight maneuvers or tracking errors when driving in close proximity to obstacles. DiTree trajectories on the other hand had no collisions.

In addition to number of collisions, we also measure tracking deviation by calculating the distance from each point during execution to the nearest point on the planned path. We find RRT shows a larger average deviation of $0.5[m]$, compared with only $0.28[m]$ for DiTree. A full recording of the experiment can be found in the supplementary video.

D.1.1 Full Results for Real-World Experiment

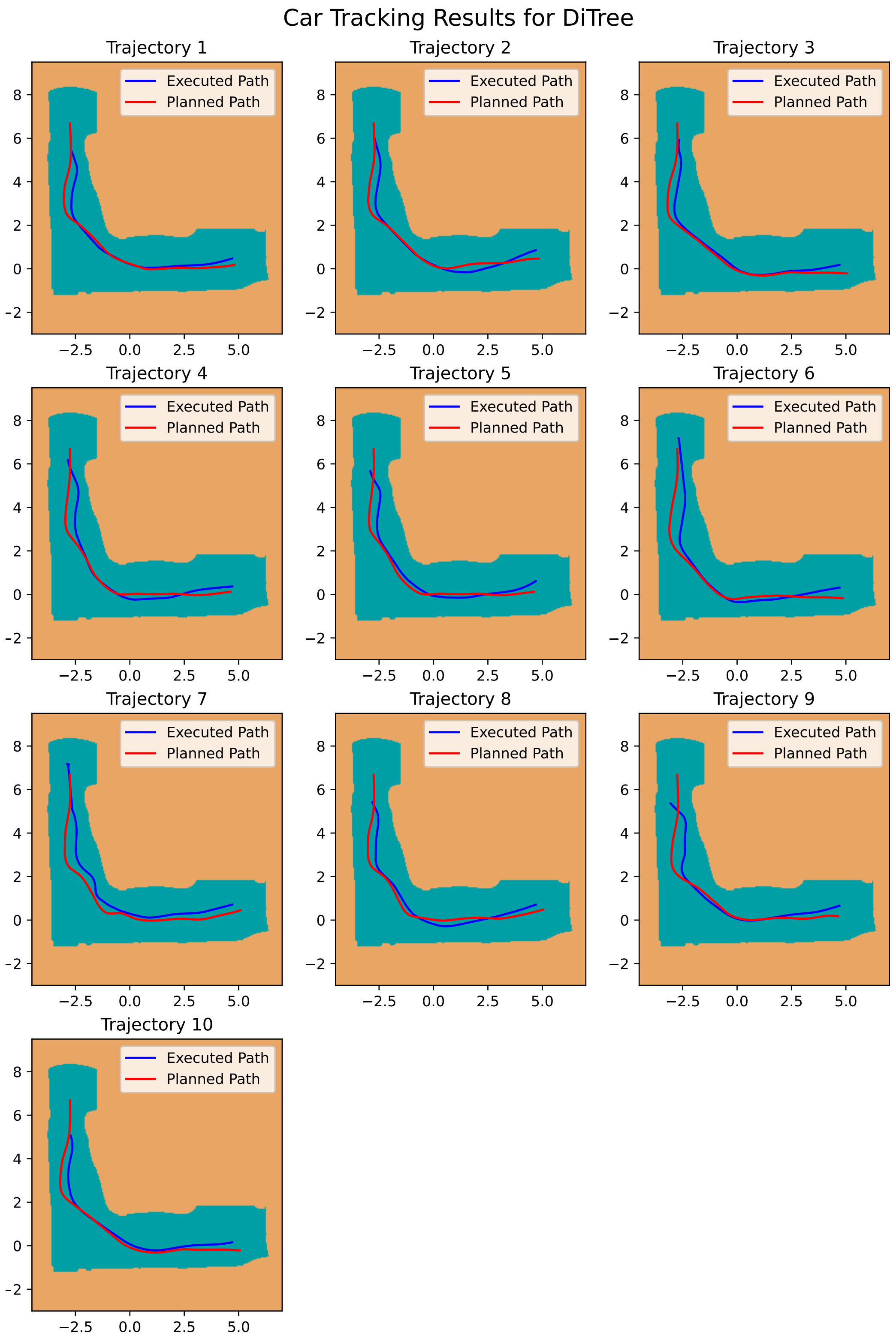


Figure 7: Real car tracking results for DiTree. All axes are expressed in meters.

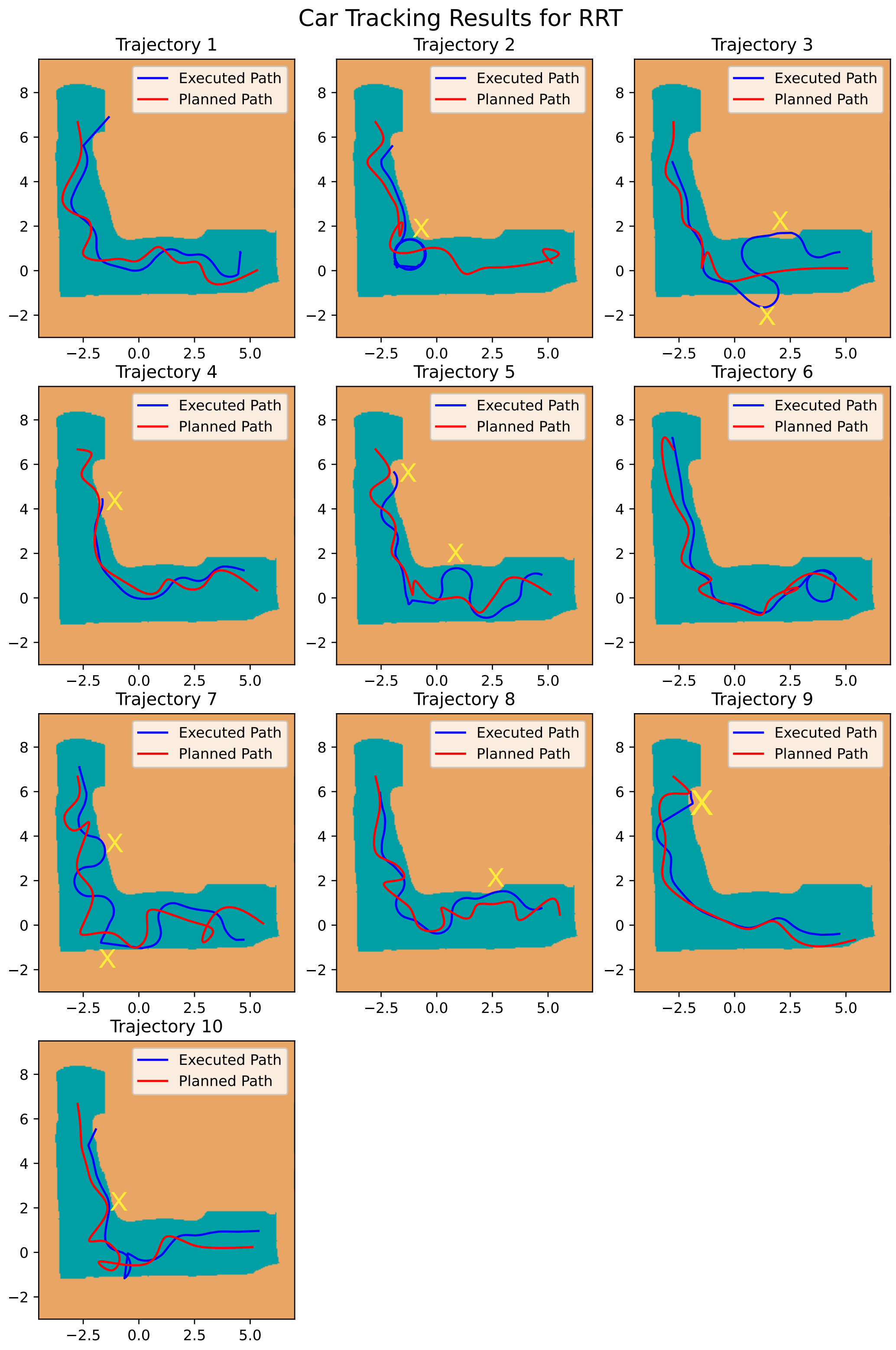


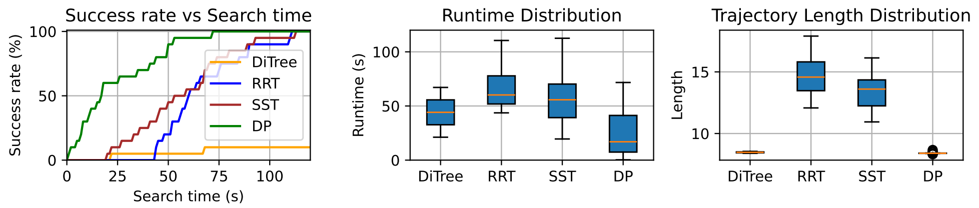
Figure 8: Real car tracking results for RRT, where collisions are marked by a yellow ’X’. Only Trajectories 1 and 6 avoid collisions during execution. Impact with an obstacle while following a loop during trajectory 2 caused a tracking issue. All axes are expressed in meters.

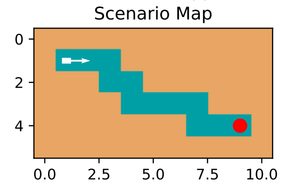
D.1.2 Full Results for Main Experiment

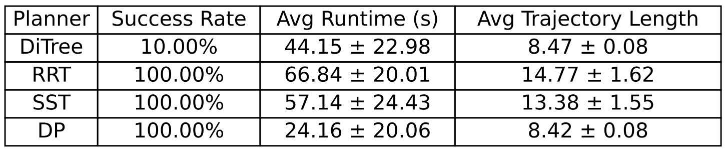
Computational Bottleneck. Traditional planners such as RRT and SST rely heavily on low-level geometric operations, issuing approximately 798.6k collision checks per minute of planning in a typical scenario. In contrast, DiTree reduces this number to just 10.5k collision checks per minute. Instead of spending computation time on frequent collision evaluations and nearest-neighbor queries, DiTree allocates the majority of its runtime—94.34%—to diffusion model inference. This represents a clear reallocation of computational resources from classical search primitives to learned, amortized inference, reflecting a distinct change in the planner's computational profile. While our bare-bones implementation already exhibits good performance, these results also suggest significant potential for further efficiency gains through model distillation, caching, and other inference-time optimizations [68] that could substantially reduce the cost of diffusion-based planning.

Below, we provide full details on the experimental results reported in the main text.

CarMaze Results for Corridor - 120 Seconds

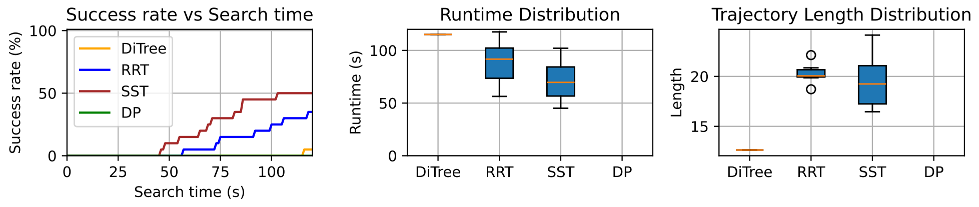


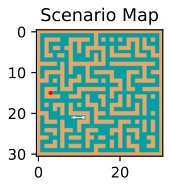


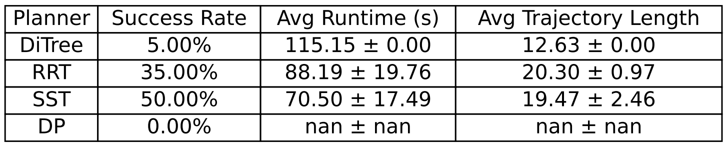


Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	10.00%	44.15 ± 22.98	8.47 ± 0.08
RRT	100.00%	66.84 ± 20.01	14.77 ± 1.62
SST	100.00%	57.14 ± 24.43	13.38 ± 1.55
DP	100.00%	24.16 ± 20.06	8.42 ± 0.08

CarMaze Results for Huge 1 - 120 Seconds

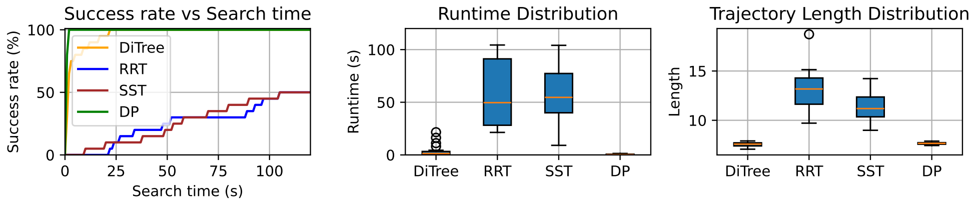


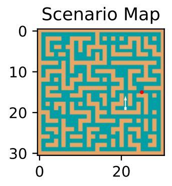




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	5.00%	115.15 ± 0.00	12.63 ± 0.00
RRT	35.00%	88.19 ± 19.76	20.30 ± 0.97
SST	50.00%	70.50 ± 17.49	19.47 ± 2.46
DP	0.00%	nan ± nan	nan ± nan

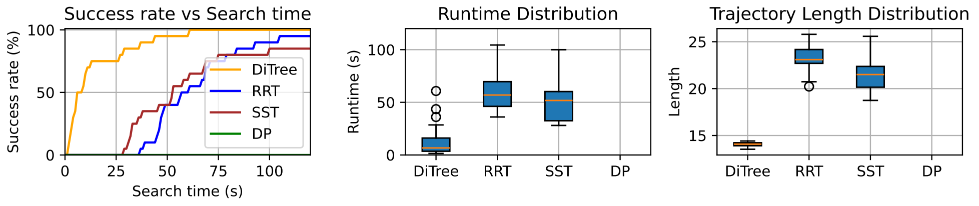
CarMaze Results for Huge 2 - 120 Seconds

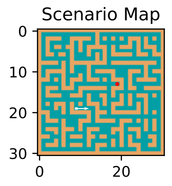




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	100.00%	4.03 ± 5.69	7.56 ± 0.21
RRT	50.00%	58.56 ± 31.56	13.29 ± 2.39
SST	50.00%	56.72 ± 28.55	11.34 ± 1.47
DP	100.00%	0.52 ± 0.41	7.66 ± 0.11

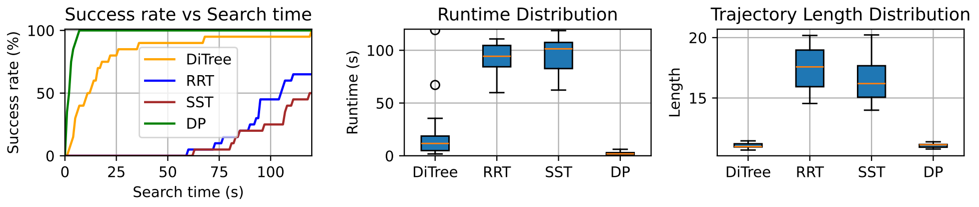
CarMaze Results for Huge 3 - 120 Seconds

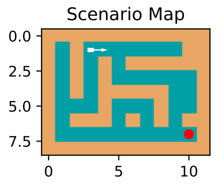




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	100.00%	14.02 ± 16.05	14.05 ± 0.25
RRT	95.00%	61.00 ± 18.35	23.29 ± 1.41
SST	85.00%	50.53 ± 18.94	21.64 ± 1.86
DP	0.00%	nan ± nan	nan ± nan

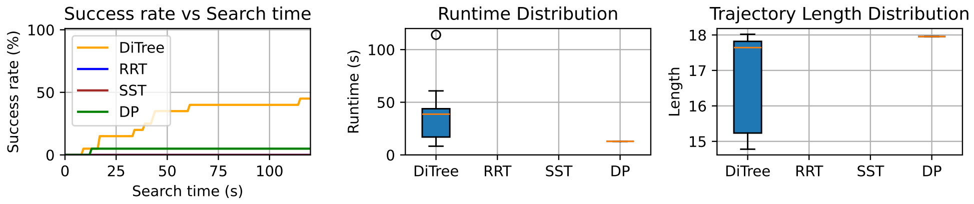
CarMaze Results for Large 1 - 120 Seconds

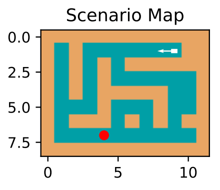




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	100.00%	19.94 ± 27.07	11.06 ± 0.19
RRT	65.00%	91.20 ± 14.27	17.53 ± 1.70
SST	50.00%	95.59 ± 16.68	16.50 ± 1.82
DP	100.00%	2.12 ± 1.74	11.07 ± 0.15

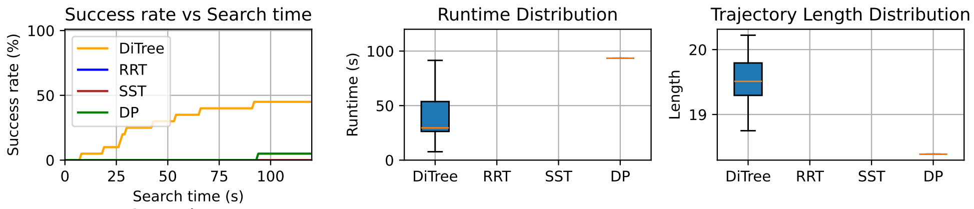
CarMaze Results for Large 2 - 120 Seconds

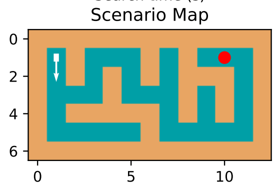




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	45.00%	41.68 ± 29.90	16.83 ± 1.31
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	5.00%	12.89 ± 0.00	17.96 ± 0.00

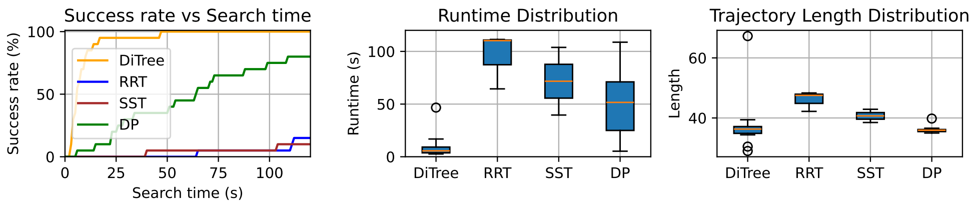
CarMaze Results for Race - 120 Seconds

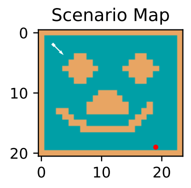




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	45.00%	40.26 ± 24.59	19.51 ± 0.42
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	5.00%	93.47 ± 0.00	18.39 ± 0.00

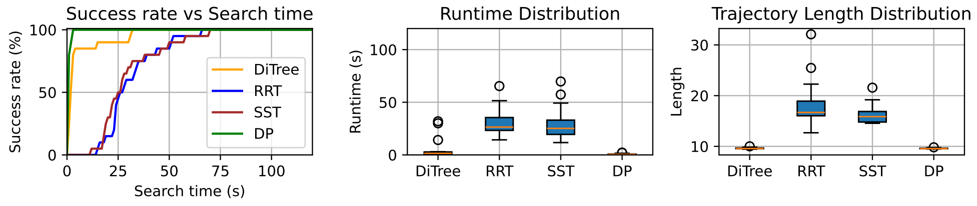
CarMaze Results for Smiley 1 - 120 Seconds

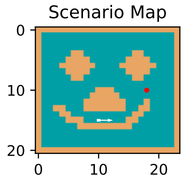




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	100.00%	8.62 ± 9.48	37.26 ± 7.29
RRT	15.00%	95.36 ± 21.86	46.01 ± 2.72
SST	10.00%	71.71 ± 32.11	40.69 ± 2.20
DP	80.00%	51.42 ± 30.22	36.06 ± 1.07

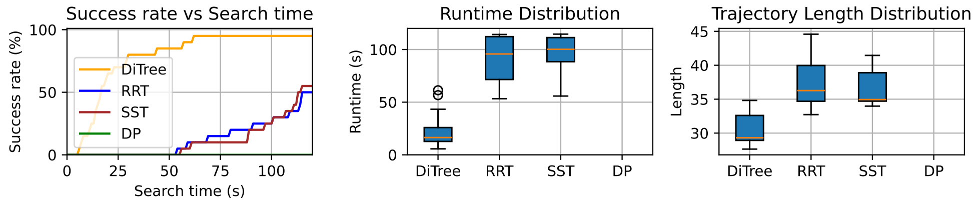
CarMaze Results for Smiley 2 - 120 Seconds

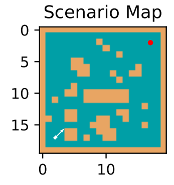




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	100.00%	5.09 ± 9.10	9.62 ± 0.14
RRT	100.00%	31.00 ± 12.77	18.13 ± 4.41
SST	100.00%	29.60 ± 14.63	16.28 ± 1.82
DP	100.00%	0.72 ± 0.58	9.59 ± 0.09

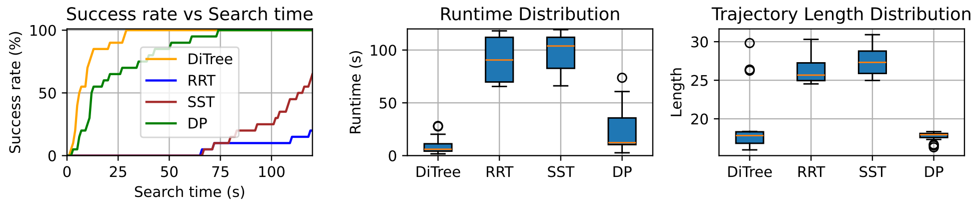
CarMaze Results for Warehouse 1 - 120 Seconds

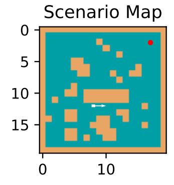




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	95.00%	21.69 ± 15.51	30.80 ± 2.42
RRT	50.00%	90.18 ± 22.44	37.31 ± 3.59
SST	55.00%	95.08 ± 19.58	36.75 ± 2.78
DP	0.00%	nan ± nan	nan ± nan

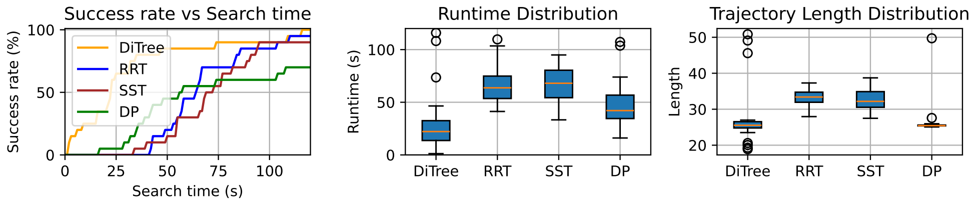
CarMaze Results for Warehouse 2 - 120 Seconds

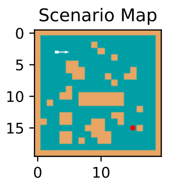




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	100.00%	9.25 ± 7.52	18.94 ± 3.72
RRT	20.00%	91.21 ± 23.14	26.54 ± 2.26
SST	65.00%	98.37 ± 17.40	27.38 ± 1.74
DP	100.00%	23.22 ± 19.82	17.72 ± 0.57

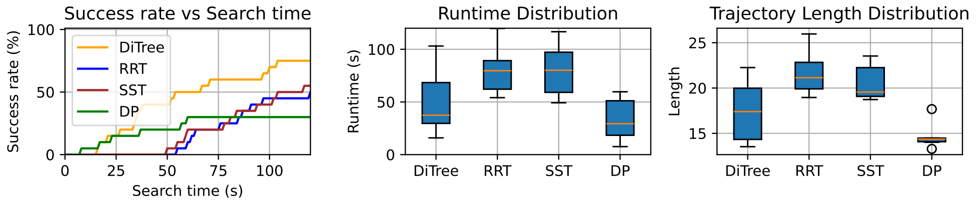
CarMaze Results for Warehouse 3 - 120 Seconds

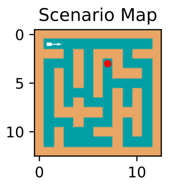




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	100.00%	31.26 ± 31.53	27.91 ± 9.03
RRT	95.00%	65.49 ± 19.00	33.18 ± 2.28
SST	90.00%	68.31 ± 17.51	32.60 ± 2.73
DP	70.00%	50.98 ± 26.06	27.32 ± 6.24

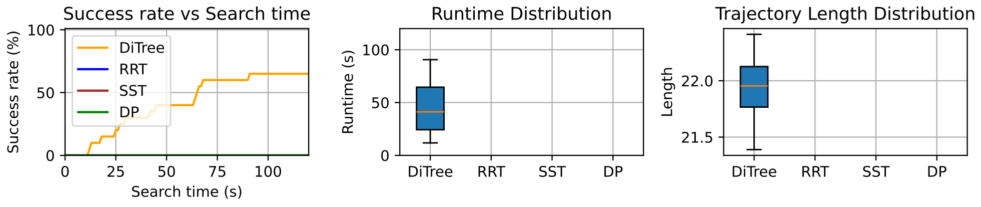
CarMaze Results for XL 1 - 120 Seconds

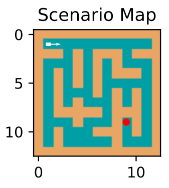




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	75.00%	50.67 ± 29.19	17.21 ± 3.28
RRT	50.00%	79.03 ± 19.39	21.81 ± 2.36
SST	55.00%	79.86 ± 21.28	20.62 ± 1.81
DP	30.00%	33.19 ± 19.43	14.70 ± 1.39

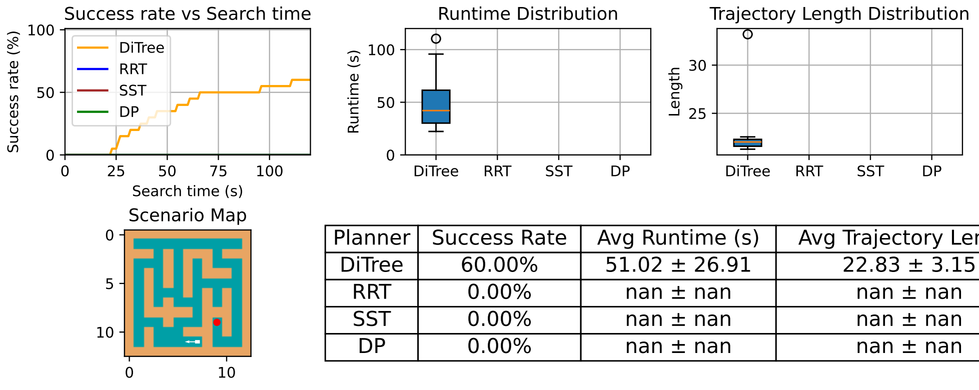
CarMaze Results for XL 2 - 120 Seconds





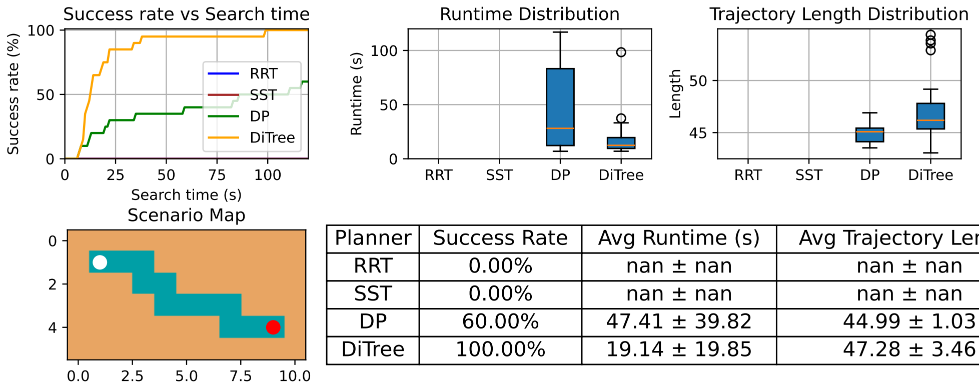
Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	65.00%	43.01 ± 24.14	21.93 ± 0.31
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	0.00%	nan ± nan	nan ± nan

CarMaze Results for XL 3 - 120 Seconds



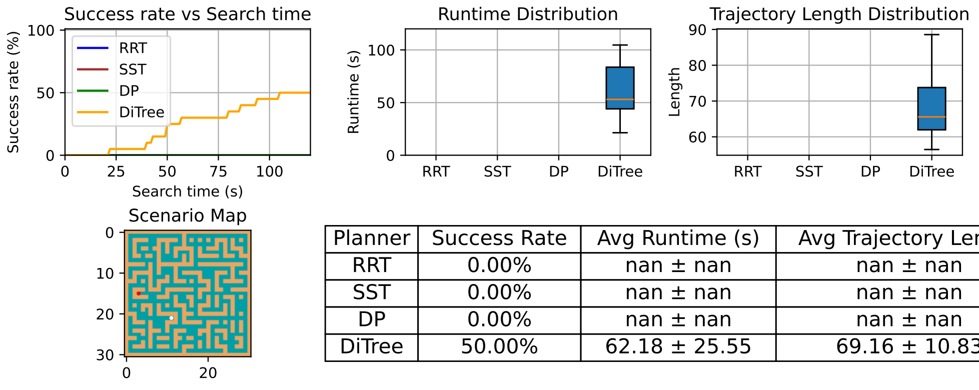
Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
DiTree	60.00%	51.02 ± 26.91	22.83 ± 3.15
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	0.00%	nan ± nan	nan ± nan

AntMaze Results for Corridor - 120 Seconds



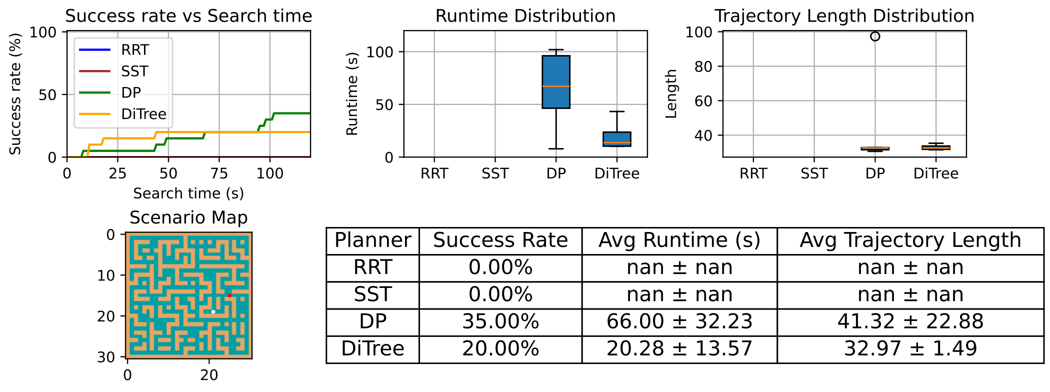
Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	60.00%	47.41 ± 39.82	44.99 ± 1.03
DiTree	100.00%	19.14 ± 19.85	47.28 ± 3.46

AntMaze Results for Huge 1 - 120 Seconds



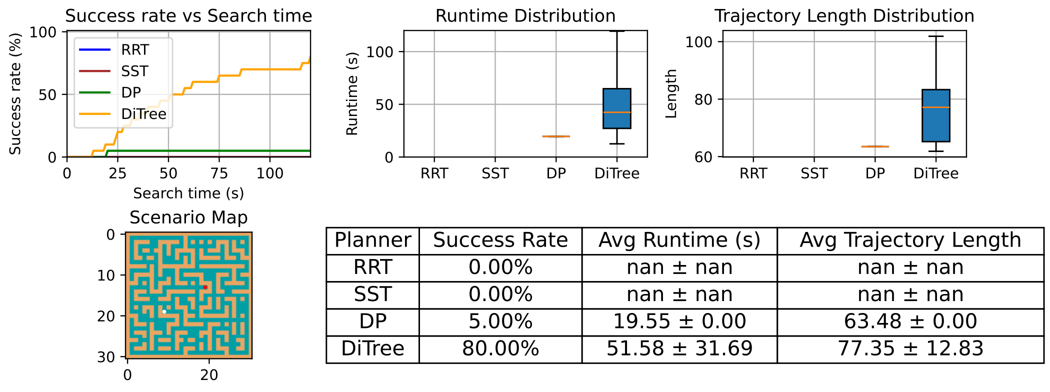
Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	0.00%	nan ± nan	nan ± nan
DiTree	50.00%	62.18 ± 25.55	69.16 ± 10.83

AntMaze Results for Huge 2 - 120 Seconds



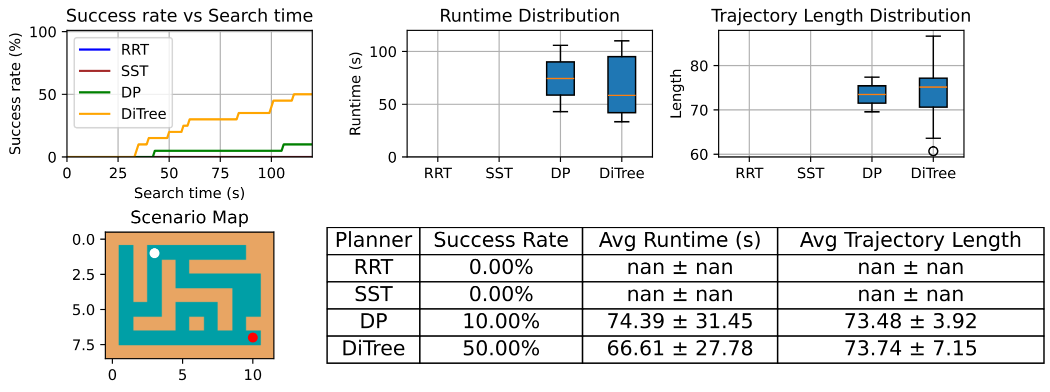
Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	35.00%	66.00 ± 32.23	41.32 ± 22.88
DiTree	20.00%	20.28 ± 13.57	32.97 ± 1.49

AntMaze Results for Huge 3 - 120 Seconds



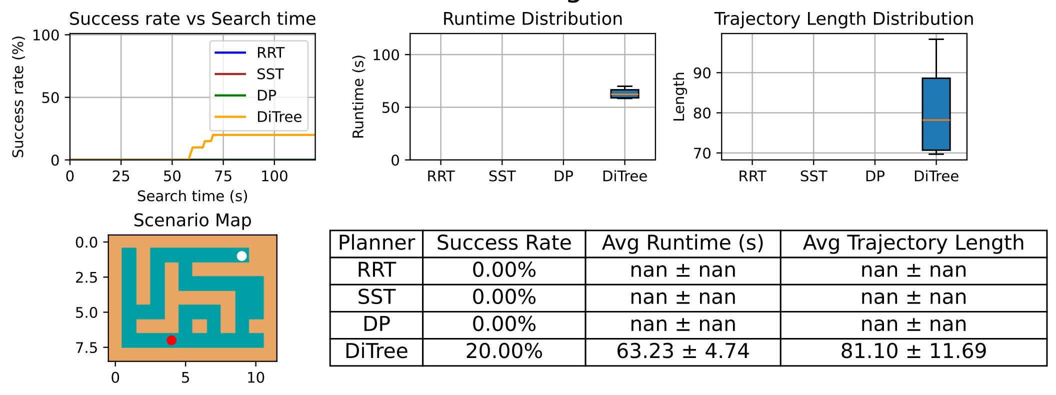
Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	5.00%	19.55 ± 0.00	63.48 ± 0.00
DiTree	80.00%	51.58 ± 31.69	77.35 ± 12.83

AntMaze Results for Large 1 - 120 Seconds



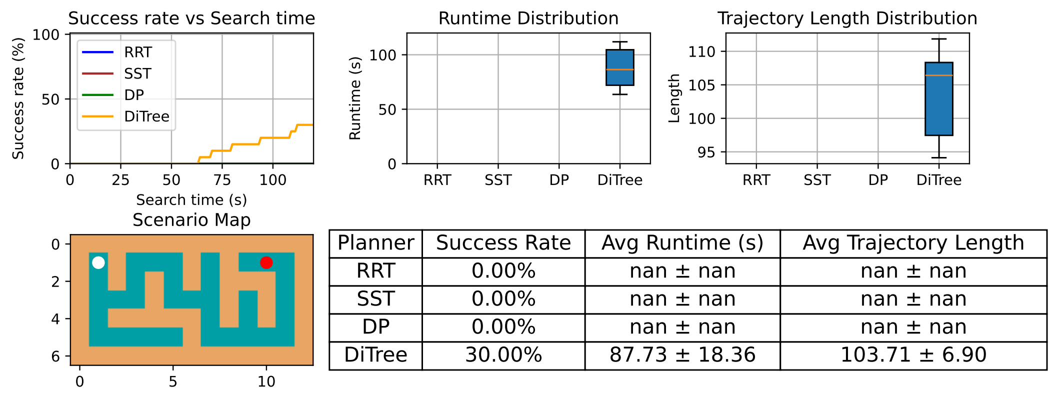
Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	10.00%	74.39 ± 31.45	73.48 ± 3.92
DiTree	50.00%	66.61 ± 27.78	73.74 ± 7.15

AntMaze Results for Large 2 - 120 Seconds



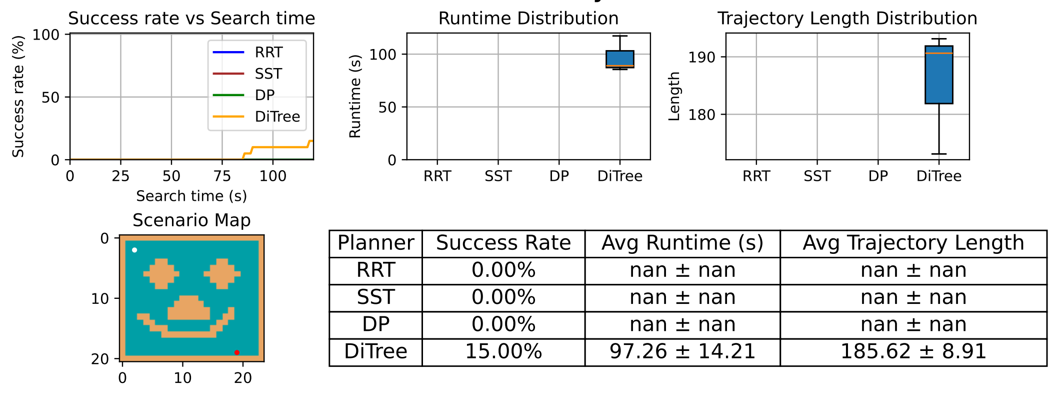
Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	0.00%	nan ± nan	nan ± nan
DiTree	20.00%	63.23 ± 4.74	81.10 ± 11.69

AntMaze Results for Race - 120 Seconds



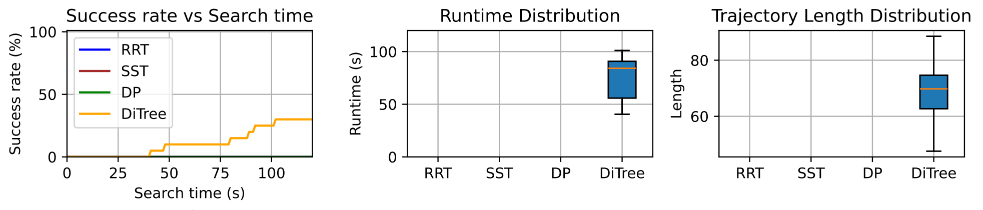
Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	0.00%	nan ± nan	nan ± nan
DiTree	30.00%	87.73 ± 18.36	103.71 ± 6.90

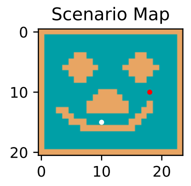
AntMaze Results for Smiley 1 - 120 Seconds



Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	0.00%	nan ± nan	nan ± nan
DiTree	15.00%	97.26 ± 14.21	185.62 ± 8.91

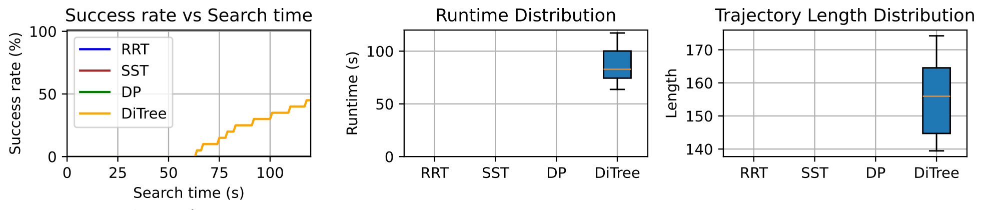
AntMaze Results for Smiley 2 - 120 Seconds

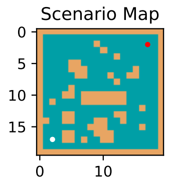




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	0.00%	nan ± nan	nan ± nan
DiTree	30.00%	74.88 ± 22.66	68.71 ± 12.63

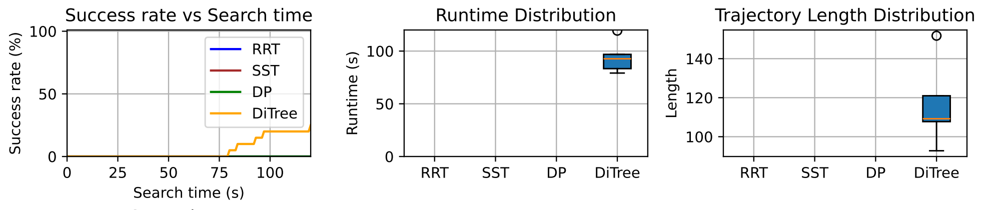
AntMaze Results for Warehouse 1 - 120 Seconds

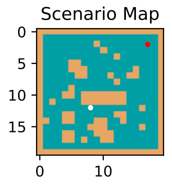




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	0.00%	nan ± nan	nan ± nan
DiTree	45.00%	87.19 ± 17.67	154.31 ± 11.82

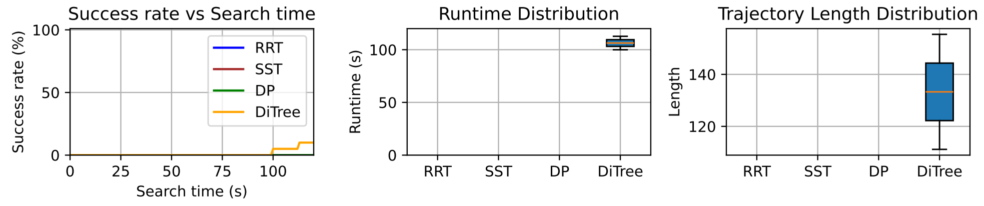
AntMaze Results for Warehouse 2 - 120 Seconds

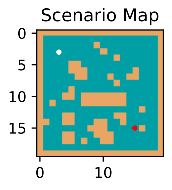




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	0.00%	nan ± nan	nan ± nan
DiTree	25.00%	94.28 ± 14.03	116.50 ± 19.77

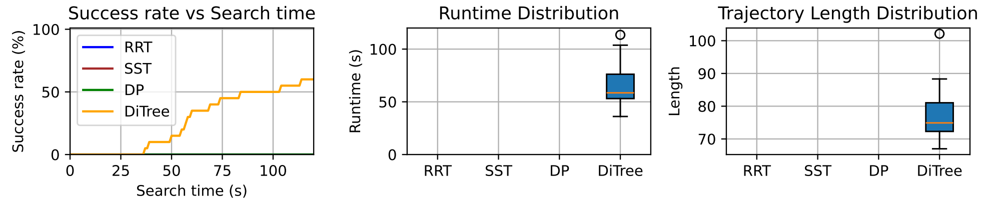
AntMaze Results for Warehouse 3 - 120 Seconds

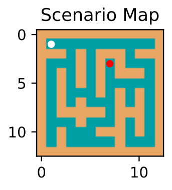




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	0.00%	nan ± nan	nan ± nan
DiTree	10.00%	106.31 ± 6.39	133.28 ± 22.12

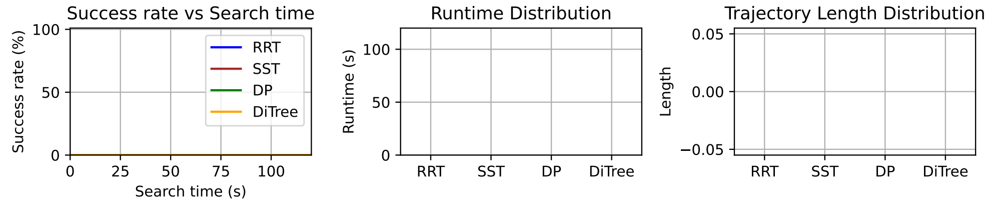
AntMaze Results for XL 1 - 120 Seconds

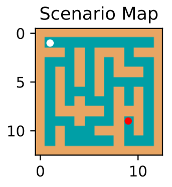




Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	0.00%	nan ± nan	nan ± nan
DiTree	60.00%	66.19 ± 22.93	78.07 ± 9.23

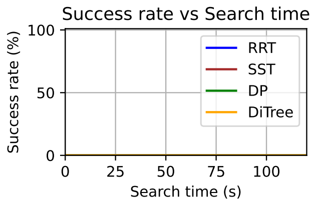
AntMaze Results for XL 2 - 120 Seconds

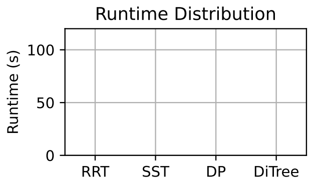


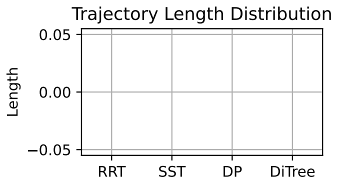


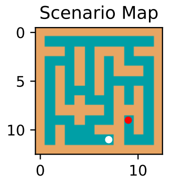
Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	0.00%	nan ± nan	nan ± nan
DiTree	0.00%	nan ± nan	nan ± nan

AntMaze Results for XL 3 - 120 Seconds









Planner	Success Rate	Avg Runtime (s)	Avg Trajectory Length
RRT	0.00%	nan ± nan	nan ± nan
SST	0.00%	nan ± nan	nan ± nan
DP	0.00%	nan ± nan	nan ± nan
DiTree	0.00%	nan ± nan	nan ± nan