Planning with Diffusion for Flexible Behavior Synthesis

Michael Janner [*1] **Yilun Du** [*2] **Joshua B. Tenenbaum** [2] **Sergey Levine** [1]

Abstract

Model-based reinforcement learning methods often use learning only for the purpose of estimating an approximate dynamics model, offloading the rest of the decision-making work to classical trajectory optimizers. While conceptually simple, this combination has a number of empirical shortcomings, suggesting that learned models may not be well-suited to standard trajectory optimization. In this paper, we consider what it would look like to fold as much of the trajectory optimization pipeline as possible into the modeling problem, such that sampling from the model and planning with it become nearly identical. The core of our technical approach lies in a diffusion probabilistic model that plans by iteratively denoising trajectories. We show how classifier-guided sampling and image inpainting can be reinterpreted as coherent planning strategies, explore the unusual and useful properties of diffusion-based planning methods, and demonstrate the effectiveness of our framework in control settings that emphasize long-horizon decision-making and test-time flexibility.

1 Introduction

Planning with a learned model is a conceptually simple framework for reinforcement learning and data-driven decision-making. Its appeal comes from employing learning techniques only where they are the most mature and effective: for the approximation of unknown environment dynamics in what amounts to a supervised learning problem. Afterwards, the learned model may be plugged into classical trajectory optimization routines (Tassa et al., 2012; Posa et al., 2014; Kelly, 2017), which are similarly well-understood in their original context.

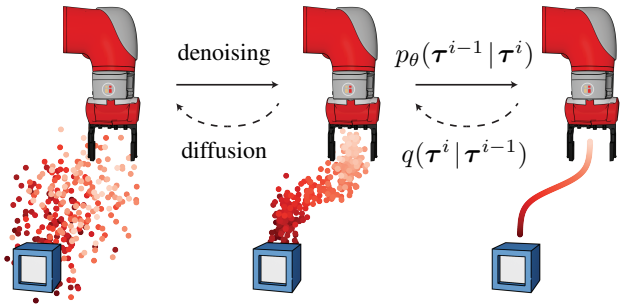

Figure 1. Diffuser is a diffusion probabilistic model that plans by iteratively refining trajectories.

However, this combination rarely works as described. Because powerful trajectory optimizers exploit learned models, plans generated by this procedure often look more like adversarial examples than optimal trajectories (Talvitie, 2014; Ke et al., 2018). As a result, contemporary model-based reinforcement learning algorithms often inherit more from model-free methods, such as value functions and policy gradients (Wang et al., 2019), than from the trajectory optimization toolbox. Those methods that do rely on online planning tend to use simple gradient-free trajectory optimization routines like random shooting (Nagabandi et al., 2018) or the cross-entropy method (Botev et al., 2013; Chua et al., 2018) to avoid the aforementioned issues.

In this work, we propose an alternative approach to data-driven trajectory optimization. The core idea is to train a model that is directly amenable to trajectory optimization, in the sense that sampling from the model and planning with it become nearly identical. This goal requires a shift in how the model is designed. Because learned dynamics models are normally meant to be proxies for environment dynamics, improvements are often achieved by structuring the model according to the underlying causal process (Bapst et al., 2019). Instead, we consider how to design a model in line with the planning problem in which it will be used. For example, because the model will ultimately be used for planning, action distributions are just as important as state dynamics and long-horizon accuracy is more important than single-step error. On the other hand, the model should remain agnostic to reward function so that it may be used

[*]Equal contribution [1]University of California, Berkeley [2]MIT. Correspondence to: janner@berkeley.edu, yilundu@mit.edu.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

Code and visualizations of the learned denoising process are available at **diffusion-planning.github.io**.

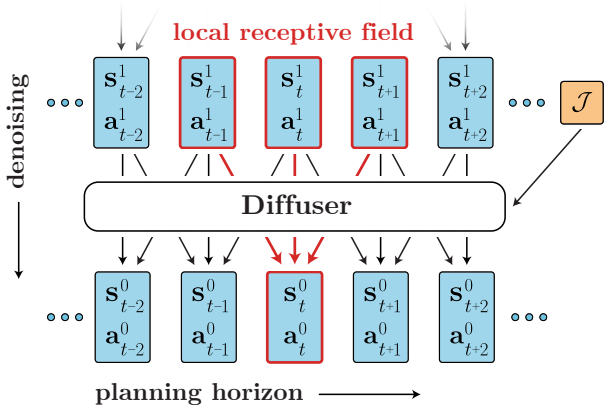

Figure 2. Diffuser samples plans by iteratively denoising two-dimensional arrays consisting of a variable number of state-action pairs. A small receptive field constrains the model to only enforce local consistency during a single denoising step. By composing many denoising steps together, local consistency can drive global coherence of a sampled plan. An optional guide function $\mathcal{J}$ can be used to bias plans toward those optimizing a test-time objective or satisfying a set of constraints.

in multiple tasks, including those unseen during training. Finally, the model should be designed so that its plans, and not just its predictions, improve with experience and are resistant to the myopic failure modes of standard shooting-based planning algorithms.

We instantiate this idea as a trajectory-level diffusion probabilistic model (Sohl-Dickstein et al., 2015; Ho et al., 2020) called Diffuser, visualized in Figure 2. Whereas standard model-based planning techniques predict forward in time autoregressively, Diffuser predicts all timesteps of a plan simultaneously. The iterative sampling process of diffusion models leads to flexible conditioning, allowing for auxiliary guides to modify the sampling procedure to recover trajectories with high return or satisfying a set of constraints. This formulation of data-driven trajectory optimization has several appealing properties:

Long-horizon scalability Diffuser is trained for the accuracy of its generated trajectories rather than its single-step error, so it does not suffer from the compounding rollout errors of single-step dynamics models and scales more gracefully with respect to long planning horizon.

Task compositionality Reward functions provide auxiliary gradients to be used while sampling a plan, allowing for a straightforward way of planning by composing multiple rewards simultaneously by adding together their gradients.

Temporal compositionality Diffuser generates globally coherent trajectories by iteratively improving local consistency, allowing it to generalize to novel trajectories by stitching together in-distribution subsequences.

Effective non-greedy planning By blurring the line between model and planner, the training procedure that improves the model's predictions also has the effect of improving its planning capabilities. This design yields a learned planner that can solve the types of long-horizon, sparse-reward problems that prove difficult for many conventional planning methods.

The core contribution of this work is a denoising diffusion model designed for trajectory data and an associated probabilistic framework for behavior synthesis. While unconventional compared to the types of models routinely used in deep model-based reinforcement learning, we demonstrate that Diffuser has a number of useful properties and is particularly effective in control settings that require long-horizon reasoning and test-time flexibility.

2 Background

Our approach to planning is a learning-based analogue of past work in behavioral synthesis using trajectory optimization (Witkin & Kass, 1988; Tassa et al., 2012). In this section, we provide a brief background on the problem setting considered by trajectory optimization and the class of generative models we employ for that problem.

2.1 Problem Setting

Consider a system governed by the discrete-time dynamics $\mathbf{s}_{t+1} = \boldsymbol{f}(\mathbf{s}_t, \mathbf{a}_t)$ at state $\mathbf{s}_t$ given an action $\mathbf{a}_t$. Trajectory optimization refers to finding a sequence of actions $\mathbf{a}^*_{0:T}$ that maximizes (or minimizes) an objective $\mathcal{J}$ factorized over per-timestep rewards (or costs) $r(\mathbf{s}_t, \mathbf{a}_t)$:

$$\mathbf{a}^*_{0:T} = \arg\max_{\mathbf{a}_{0:T}} \mathcal{J}(\mathbf{s}_0, \mathbf{a}_{0:T}) = \arg\max_{\mathbf{a}_{0:T}} \sum_{t=0}^{T} r(\mathbf{s}_t, \mathbf{a}_t)$$

where T is the planning horizon. We use the abbreviation $\boldsymbol{\tau} = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)$ to refer to a trajectory of interleaved states and actions and $\mathcal{J}(\boldsymbol{\tau})$ to denote the objective value of that trajectory.

2.2 Diffusion Probabilistic Models

Diffusion probabilistic models (Sohl-Dickstein et al., 2015; Ho et al., 2020) pose the data-generating process as an iterative denoising procedure $p_\theta(\boldsymbol{\tau}^{i-1} \mid \boldsymbol{\tau}^i)$. This denoising is the reverse of a forward diffusion process $q(\boldsymbol{\tau}^i \mid \boldsymbol{\tau}^{i-1})$ that slowly corrupts the structure in data by adding noise. The data distribution induced by the model is given by:

$$p_\theta(\boldsymbol{\tau}^0) = \int p(\boldsymbol{\tau}^N) \prod_{i=1}^{N} p_\theta(\boldsymbol{\tau}^{i-1} \mid \boldsymbol{\tau}^i) \mathrm{d}\boldsymbol{\tau}^{1:N}$$

where $p(\boldsymbol{\tau}^N)$ is a standard Gaussian prior and $\boldsymbol{\tau}^0$ denotes (noiseless) data. Parameters θ are optimized by minimizing

a variational bound on the negative log likelihood of the reverse process: $\theta^* = \arg\min_\theta -\mathbb{E}_{\boldsymbol{\tau}^0}\left[\log p_\theta(\boldsymbol{\tau}^0)\right]$. The reverse process is often parameterized as Gaussian with fixed timestep-dependent covariances:

$$p_\theta(\boldsymbol{\tau}^{i-1} \mid \boldsymbol{\tau}^i) = \mathcal{N}(\boldsymbol{\tau}^{i-1} \mid \mu_\theta(\boldsymbol{\tau}^i, i), \Sigma^i).$$

The forward process $q(\boldsymbol{\tau}^i \mid \boldsymbol{\tau}^{i-1})$ is typically prespecified.

Notation. There are two "times" at play in this work: that of the diffusion process and that of the planning problem. We use superscripts (i when unspecified) to denote diffusion timestep and subscripts (t when unspecified) to denote planning timestep. For example, $\mathbf{s}_t^0$ refers to the t^{th} state in a noiseless trajectory. When it is unambiguous from context, superscripts of noiseless quantities are omitted: $\boldsymbol{\tau} = \boldsymbol{\tau}^0$. We overload notation slightly by referring to the t^{th} state (or action) in a trajectory $\boldsymbol{\tau}$ as $\boldsymbol{\tau}_{\mathbf{s}_t}$ (or $\boldsymbol{\tau}_{\mathbf{a}_t}$).

3 Planning with Diffusion

A major obstacle to using trajectory optimization techniques is that they require knowledge of the environment dynamics $\boldsymbol{f}$. Most learning-based methods attempt to overcome this obstacle by training an approximate dynamics model and plugging it in to a conventional planning routine. However, learned models are often poorly suited to the types of planning algorithms designed with ground-truth models in mind, leading to planners that exploit learned models by finding adversarial examples.

We propose a tighter coupling between modeling and planning. Instead of using a learned model in the context of a classical planner, we subsume as much of the planning process as possible into the generative modeling framework, such that planning becomes nearly identical to sampling. We do this using a diffusion model of trajectories, $p_\theta(\boldsymbol{\tau})$. The iterative denoising process of a diffusion model lends itself to flexible conditioning by way of sampling from perturbed distributions of the form:

$$\tilde{p}_\theta(\boldsymbol{\tau}) \propto p_\theta(\boldsymbol{\tau}) h(\boldsymbol{\tau}). \tag{1}$$

The function $h(\boldsymbol{\tau})$ can contain information about prior evidence (such as an observation history), desired outcomes (such as a goal to reach), or general functions to optimize (such as rewards or costs). Performing inference in this perturbed distribution can be seen as a probabilistic analogue to the trajectory optimization problem posed in Section 2.1, as it requires finding trajectories that are both physically realistic under $p_\theta(\boldsymbol{\tau})$ and high-reward (or constraint-satisfying) under $h(\boldsymbol{\tau})$. Because the dynamics information is separated from the perturbation distribution $h(\boldsymbol{\tau})$, a single diffusion model $p_\theta(\boldsymbol{\tau})$ may be reused for multiple tasks in the same environment.

In this section, we describe Diffuser, a diffusion model designed for learned trajectory optimization. We then discuss two specific instantiations of planning with Diffuser, realized as reinforcement learning counterparts to classifier-guided sampling and image inpainting.

3.1 A Generative Model for Trajectory Planning

Temporal ordering. Blurring the line between sampling from a trajectory model and planning with it yields an unusual constraint: we can no longer predict states autoregressively in temporal order. Consider the goal-conditioned inference $p(\mathbf{s}_1 \mid \mathbf{s}_0, \mathbf{s}_T)$; the next state $\mathbf{s}_1$ depends on a *future* state as well as a prior one. This example is an instance of a more general principle: while dynamics prediction is causal, in the sense that the present is determined by the past, decision-making and control can be anti-causal, in the sense that decisions in the present are conditional on the future.[1] Because we cannot use a temporal autoregressive ordering, we design Diffuser to predict all timesteps of a plan concurrently.

Temporal locality. Despite not being autoregressive or Markovian, Diffuser features a relaxed form of temporal locality. In Figure 2, we depict a dependency graph for a diffusion model consisting of a single temporal convolution. The receptive field of a given prediction only consists of nearby timesteps, both in the past and the future. As a result, each step of the denoising process can only make predictions based on local consistency of the trajectory. By composing many of these denoising steps together, however, local consistency can drive global coherence.

Trajectory representation. Diffuser is a model of trajectories designed for planning, meaning that the effectiveness of the controller derived from the model is just as important as the quality of the state predictions. As a result, states and actions in a trajectory are predicted jointly; for the purposes of prediction the actions are simply additional dimensions of the state. Specifically, we represent inputs (and outputs) of Diffuser as a two-dimensional array:

$$\boldsymbol{\tau} = \begin{bmatrix} \mathbf{s}_0 & \mathbf{s}_1 & \cdots & \mathbf{s}_T \\ \mathbf{a}_0 & \mathbf{a}_1 & & \mathbf{a}_T \end{bmatrix}. \tag{2}$$

with one column per timestep of the planning horizon.

Architecture. We now have the ingredients needed to specify a Diffuser architecture: **(1)** an entire trajectory should be predicted non-autoregressively, **(2)** each step of the denoising process should be temporally local,

[1] In general reinforcement learning contexts, conditioning on the future emerges from the assumption of future optimality for the purpose of writing a dynamic programming recursion. Concretely, this appears as the future optimality variables $\mathcal{O}_{t:T}$ in the action distribution $\log p(\mathbf{a}_t \mid \mathbf{s}_t, \mathcal{O}_{t:T})$ (Levine, 2018).

Algorithm 1 Guided Diffusion Planning

Require Diffuser μ_θ, guide $\mathcal{J}$, scale α, covariances Σ^i
while not done **do**
  Observe state $\mathbf{s}$; initialize plan $\boldsymbol{\tau}^N \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  for $i = N, \dots, 1$ **do**
    // parameters of reverse transition
    $\mu \leftarrow \mu_\theta(\boldsymbol{\tau}^i)$
    // guide using gradients of return
    $\boldsymbol{\tau}^{i-1} \sim \mathcal{N}(\mu + \alpha \Sigma \nabla \mathcal{J}(\mu), \Sigma^i)$
    // constrain first state of plan
    $\boldsymbol{\tau}^{i-1}_{\boldsymbol{s}_0} \leftarrow \boldsymbol{s}$
  Execute first action of plan $\boldsymbol{\tau}^0_{\mathbf{a}_0}$

and **(3)** the trajectory representation should allow for equivariance along one dimension (the planning horizon) but not the other (the state and action features). We satisfy these criteria with a model consisting of repeated (temporal) convolutional residual blocks. The overall architecture resembles the types of U-Nets that have found success in image-based diffusion models, but with two-dimensional spatial convolutions replaced by one-dimensional temporal convolutions (Figure A1). Because the model is fully convolutional, the horizon of the predictions is determined not by the model architecture, but by the input dimensionality; it can change dynamically during planning if desired.

Training. We use Diffuser to parameterize a learned gradient $\epsilon_\theta(\boldsymbol{\tau}^i, i)$ of the trajectory denoising process, from which the mean μ_θ can be solved in closed form (Ho et al., 2020). We use the simplified objective for training the ϵ-model, given by:

$$\mathcal{L}(\theta) = \mathbb{E}_{i, \epsilon, \boldsymbol{\tau}^0} \left[\|\epsilon - \epsilon_\theta(\boldsymbol{\tau}^i, i)\|^2 \right],$$

in which $i \sim \mathcal{U}\{1, 2, \dots, N\}$ is the diffusion timestep, $\epsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is the noise target, and $\boldsymbol{\tau}^i$ is the trajectory $\boldsymbol{\tau}^0$ corrupted with noise ϵ. Reverse process covariances Σ^i follow the cosine schedule of Nichol & Dhariwal (2021).

3.2 Reinforcement Learning as Guided Sampling

In order to solve reinforcement learning problems with Diffuser, we must introduce a notion of reward. We appeal to the control-as-inference graphical model (Levine, 2018) to do so. Let $\mathcal{O}_t$ be a binary random variable denoting the optimality of timestep t of a trajectory, with $p(\mathcal{O}_t = 1) = \exp(r(\mathbf{s}_t, \mathbf{a}_t))$. We can sample from the set of optimal trajectories by setting $h(\boldsymbol{\tau}) = p(\mathcal{O}_{1:T} \mid \boldsymbol{\tau})$ in Equation 1:

$$\tilde{p}_\theta(\boldsymbol{\tau}) = p(\boldsymbol{\tau} \mid \mathcal{O}_{1:T} = 1) \propto p(\boldsymbol{\tau}) p(\mathcal{O}_{1:T} = 1 \mid \boldsymbol{\tau}).$$

We have exchanged the reinforcement learning problem for one of *conditional sampling*. Thankfully, there has been much prior work on conditional sampling with diffusion models. While it is intractable to sample from this distribution exactly, when $p(\mathcal{O}_{1:T} \mid \boldsymbol{\tau}^i)$ is sufficiently smooth, the reverse diffusion process transitions can be approximated as Gaussian (Sohl-Dickstein et al., 2015):

$$p_\theta(\boldsymbol{\tau}^{i-1} \mid \boldsymbol{\tau}^i, \mathcal{O}_{1:T}) \approx \mathcal{N}(\boldsymbol{\tau}^{i-1}; \mu + \Sigma g, \Sigma) \tag{3}$$

where μ, Σ are the parameters of the original reverse process transition $p_\theta(\boldsymbol{\tau}^{i-1} \mid \boldsymbol{\tau}^i)$ and

$$\begin{aligned} g &= \nabla_{\boldsymbol{\tau}} \log p(\mathcal{O}_{1:T} \mid \boldsymbol{\tau})|_{\boldsymbol{\tau}=\mu} \\ &= \sum_{t=0}^{T} \nabla_{\mathbf{s}_t, \mathbf{a}_t} r(\mathbf{s}_t, \mathbf{a}_t)|_{(\mathbf{s}_t, \mathbf{a}_t)=\mu_t} = \nabla \mathcal{J}(\mu). \end{aligned}$$

This relation provides a straightforward translation between classifier-guided sampling, used to generate class-conditional images (Dhariwal & Nichol, 2021), and the reinforcement learning problem setting. We first train a diffusion model $p_\theta(\boldsymbol{\tau})$ on the states and actions of all available trajectory data. We then train a separate model $\mathcal{J}_\phi$ to predict the cumulative rewards of trajectory samples $\boldsymbol{\tau}^i$. The gradients of $\mathcal{J}_\phi$ are used to guide the trajectory sampling procedure by modifying the means μ of the reverse process according to Equation 3. The first action of a sampled trajectory $\boldsymbol{\tau} \sim p(\boldsymbol{\tau} \mid \mathcal{O}_{1:T} = 1)$ may be executed in the environment, after which the planning procedure begins again in a standard receding-horizon control loop. Pseudocode for the guided planning method is given in Algorithm 1.

3.3 Goal-Conditioned RL as Inpainting

Some planning problems are more naturally posed as constraint satisfaction than reward maximization. In these settings, the objective is to produce any feasible trajectory that satisfies a set of constraints, such as terminating at a goal location. Appealing to the two-dimensional array representation of trajectories described by Equation 2, this setting can be translated into an *inpainting problem*, in which state and action constraints act analogously to observed pixels in an image (Sohl-Dickstein et al., 2015). All unobserved locations in the array must be filled in by the diffusion model in a manner consistent with the observed constraints.

The perturbation function required for this task is a Dirac delta for observed values and constant elsewhere. Concretely, if $\mathbf{c}_t$ is state constraint at timestep t, then

$$h(\boldsymbol{\tau}) = \delta_{\mathbf{c}_t}(\mathbf{s}_0, \mathbf{a}_0, \dots, \mathbf{s}_T, \mathbf{a}_T) = \begin{cases} +\infty & \text{if } \mathbf{c}_t = \mathbf{s}_t \\ 0 & \text{otherwise} \end{cases}$$

The definition for action constraints is identical. In practice, this may be implemented by sampling from

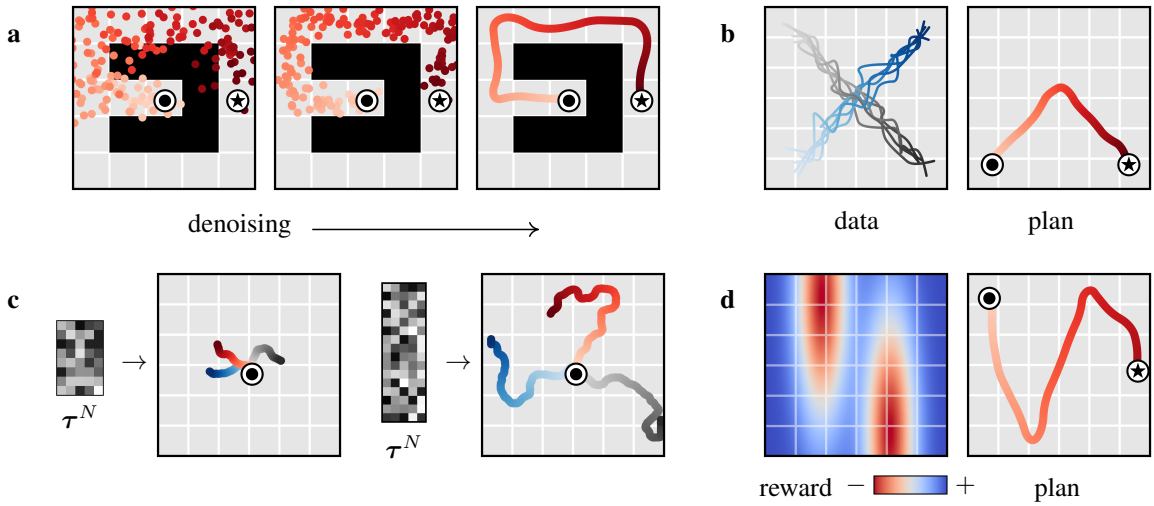

Figure 3. **(Properties of diffusion planners) (a) Learned long-horizon planning:** Diffuser's learned planning procedure does not suffer from the myopic failure modes common to shooting algorithms and is able to plan over long horizons with sparse reward. **(b) Temporal compositionality:** Even though the model is not Markovian, it generates trajectories via iterated refinements to local consistency. As a result, it exhibits the types of generalization usually associated with Markovian models, with the ability to stitch together snippets of trajectories from the training data to generate novel plan. **(c) Variable-length plans:** Despite being a trajectory-level model, Diffuser's planning horizon is not determined by its architecture. The horizon can be updated after training by changing the dimensionality of the input noise. **(d) Task compositionality:** Diffuser can be composed with new reward functions to plan for tasks unseen during training. In all subfigures, ⦿ denotes a starting state and ⍟ denotes a goal state.

the unperturbed reverse process $\boldsymbol{\tau}^{i-1} \sim p_\theta(\boldsymbol{\tau}^{i-1} \mid \boldsymbol{\tau}^i)$ and replacing the sampled values with conditioning values $\mathbf{c}_t$ after all diffusion timesteps $i \in \{0, 1, \dots, N\}$.

Even reward maximization problems require conditioning-by-inpainting because all sampled trajectories should begin at the current state. This conditioning is described by line 10 in Algorithm 1.

4 Properties of Diffusion Planners

We discuss a number of Diffuser's important properties, focusing on those that are are either distinct from standard dynamics models or unusual for non-autoregressive trajectory prediction.

Learned long-horizon planning. Single-step models are typically used as proxies for ground-truth environment dynamics $\boldsymbol{f}$, and as such are not tied to any planning algorithm in particular. In contrast, the planning routine in Algorithm 1 is closely tied to the specific affordances of diffusion models. Because our planning method is nearly identical to sampling (with the only difference being guidance by a perturbation function $h(\boldsymbol{\tau})$), Diffuser's effectiveness as a long-horizon predictor directly translates to effective long-horizon planning. We demonstrate the benefits of learned planning in a goal-reaching setting in **Figure 3a**, showing that Diffuser is able to generate feasible trajectories in the types of sparse reward settings where shooting-based approaches are known to struggle. We explore a more quantitative version of this problem setting in Section 5.1.

Temporal compositionality. Single-step models are often motivated using the Markov property, allowing them to compose in-distribution transitions to generalize to out-of-distribution trajectories. Because Diffuser generates globally coherent trajectories by iteratively improving local consistency (Section 3.1), it can also stitch together familiar subsequences in novel ways. In **Figure 3b**, we train Diffuser on trajectories that only travel in a straight line, and show that it can generalize to v-shaped trajectories by composing trajectories at their point of intersection.

Variable-length plans. Because our model is fully convolutional in the horizon dimension of its prediction, its planning horizon is not specified by architectural choices. Instead, it is determined by the size of the input noise $\boldsymbol{\tau}^N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ that initializes the denoising process, allowing for variable-length plans (**Figure 3c**).

Task compositionality. While Diffuser contains information about both environment dynamics and behaviors, it is independent of reward function. Because the model acts as a prior over possible futures, planning can be guided by comparatively lightweight perturbation functions $h(\boldsymbol{\tau})$ (or even combinations of multiple perturbations) corresponding to different rewards. We demonstrate this by planning for a new reward function unseen during training of the diffusion model (**Figure 3d**).

Environment		BCQ	CQL	IQL	Diffuser
Maze2D	U-Maze	12.8	5.7	47.4	**113.9** ±3.1
Maze2D	Medium	8.3	5.0	34.9	**121.5** ±2.7
Maze2D	Large	6.2	12.5	58.6	**123.0** ±6.4
Single-task Average		9.1	7.7	47.0	**119.5**
Multi2D	U-Maze	-	-	24.8	**128.9** ±1.8
Multi2D	Medium	-	-	12.1	**127.2** ±3.4
Multi2D	Large	-	-	13.9	**132.1** ±5.8
Multi-task Average		-	-	16.9	**129.4**

Table 1. **(Long-horizon planning)** The performance of Diffuser and prior model-free algorithms in the Maze2D environment, which tests long-horizon planning due to its sparse reward structure. The Multi2D setting refers to a multi-task variant with goal locations resampled at the beginning of every episode. Diffuser substantially outperforms prior approaches in both settings. Appendix A details the sources for the scores of the baseline algorithms.

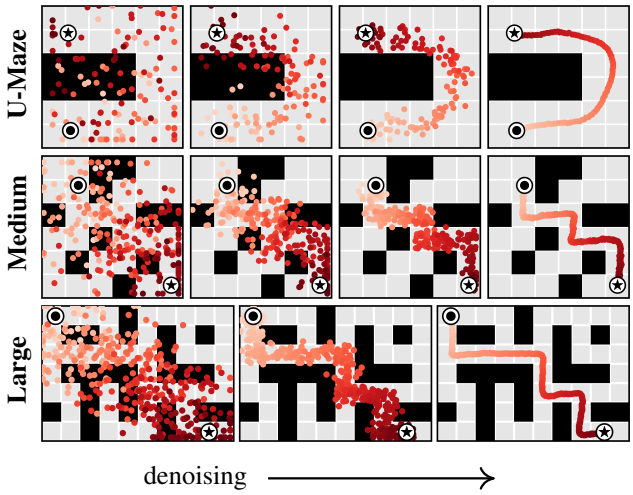

Figure 4. **(Planning as inpainting)** Plans are generated in the Maze2D environment by sampling trajectories consistent with a specified start ◉ and goal ⊛ condition. The remaining states are "inpainted" by the denoising process.

5 Experimental Evaluation

The focus of our experiments is to evaluate Diffuser on the capabilities we would like from a data-driven planner. In particular, we evaluate **(1)** the ability to plan over long horizons without manual reward shaping, **(2)** the ability to generalize to new configurations of goals unseen during training, and **(3)** the ability to recover an effective controller from heterogeneous data of varying quality. We conclude by studying practical runtime considerations of diffusion-based planning, including the most effective ways of speeding up the planning procedure while suffering minimally in terms of performance.

5.1 Long Horizon Multi-Task Planning

We evaluate long-horizon planning in the Maze2D environments (Fu et al., 2020), which require traversing to a goal location where a reward of 1 is given. No reward shaping is provided at any other location. Because it can take hundreds of steps to reach the goal location, even the best model-free algorithms struggle to adequately perform credit assignment and reliably reach the goal (Table 1).

We plan with Diffuser using the inpainting strategy to condition on a start and goal location. (The goal location is also available to the model-free methods; it is identifiable by being the only state in the dataset with non-zero reward.) We then use the sampled trajectory as an open-loop plan. Diffuser achieves scores over 100 in all maze sizes, indicating that it outperforms a reference expert policy. We visualize the reverse diffusion process generating Diffuser's plans in Figure 4.

While the training data in Maze2D is undirected – consisting of a controller navigating to and from randomly selected locations – the evaluation is single-task in that the goal is always the same. In order to test multi-task flexibility, we modify the environment to randomize the goal location at the beginning of each episode. This setting is denoted as Multi2D in Table 1. Diffuser is naturally a multi-task planner; we do not need to retrain the model from the single-task experiments and simply change the conditioning goal. As a result, Diffuser performs as well in the multi-task setting as in the single-task setting. In contrast, there is a substantial performance drop of the best model-free algorithm in the single-task setting (IQL; Kostrikov et al. 2022) when adapted to the multi-task setting. Details of our multi-task IQL with hindsight experience relabeling (Andrychowicz et al., 2017) are provided in Appendix A.

5.2 Test-time Flexibility

In order to evaluate the ability to generalize to new test-time goals, we construct a suite of block stacking tasks with three settings: **(1)** *Unconditional Stacking*, for which the task is to build a block tower as tall as possible; **(2)** *Conditional Stacking*, for which the task is to construct a block tower with a specified order of blocks, and **(3)** *Rearrangement*, for which the task is to match a set of reference blocks' locations in a novel arrangement. We train all methods on 10000 trajectories from demonstrations generated by PDDLStream (Garrett et al., 2020); rewards are equal to one upon successful stack placements and zero otherwise. These block stacking are challenging diagnostics of test-time flexibility; in the course of executing a partial stack for a randomized goal, a controller will venture into novel states not included in the training configuration.

We use one trained Diffuser for all block-stacking tasks, only

Environment	BCQ	CQL	Diffuser
Unconditional Stacking	0.0	24.4	**58.7** ±2.5
Conditional Stacking	0.0	0.0	**45.6** ±3.1
Rearrangement	0.0	0.0	**58.9** ±3.4
Average	0.0	8.9	**54.4**

Table 3. **(Test-time flexibility)** Performance of BCQ, CQL, and Diffuser on block stacking tasks. A score of 100 corresponds to a perfectly executed stack; 0 is that of a random policy.

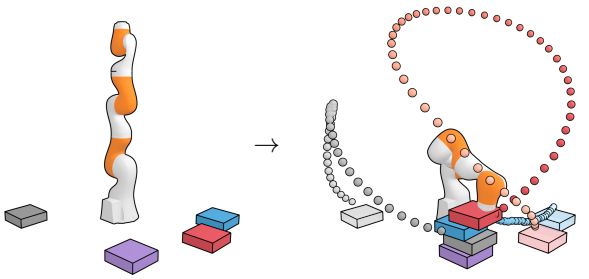

Figure 5. **(Block stacking)** A block stacking sequence executed by Diffuser. This task is best illustrated by videos viewable at `diffusion-planning.github.io`.

modifying the perturbation function $h(\boldsymbol{\tau})$ between settings. In the *Unconditional Stacking* task, we directly sample from the unperturbed denoising process $p_\theta(\boldsymbol{\tau})$ to emulate the PDDLStream controller. In the *Conditional Stacking* and *Rearrangement* tasks, we compose two perturbation functions $h(\boldsymbol{\tau})$ to bias the sampled trajectories: the first maximizes the likelihood of the trajectory's final state matching the goal configuration, and the second enforces a contact constraint between the end effector and a cube during stacking motions. (See Appendix B for details.)

We compare with two prior model-free offline reinforcement learning algorithms: BCQ (Fujimoto et al., 2019) and CQL (Kumar et al., 2020), training standard variants for *Unconditional Stacking* and goal-conditioned variants for *Conditional Stacking* and *Rearrangement*. (Baseline details are provided in Appendix A.) Quantitative results are given in Table 3, in which a score of 100 corresponds to a perfect execution of the task. Diffuser substantially outperforms both prior methods, with the conditional settings requiring flexible behavior generation proving especially difficult for the model-free algorithms. A visual depiction of an execution by Diffuser is provided in Figure 5.

5.3 Offline Reinforcement Learning

Finally, we evaluate the capacity to recover an effective single-task controller from heterogeneous data of varying quality using the D4RL offline locomotion suite (Fu et al., 2020). We guide the trajectories generated by Diffuser toward high-reward regions using the sampling procedure described in Section 3.2 and condition the trajectories on the current state using the inpainting procedure described in Section 3.3. The reward predictor $\mathcal{J}_\phi$ is trained on the same trajectories as the diffusion model.

We compare to a variety of prior algorithms spanning other approaches to data-driven control, including the model-free reinforcement learning algorithms CQL (Kumar et al., 2020), IQL (Kostrikov et al., 2022), and AWAC (Nair et al., 2020); return-conditioning approaches like Decision Transformer (DT; Chen et al. 2021b); and model-based reinforcement learning approaches including Trajectory Transformer (TT; Janner et al. 2021), MOReL (Kidambi et al., 2020), and MBOP (Argenson & Dulac-Arnold, 2021). In the single-task setting, Diffuser performance comparably to prior algorithms: better than the model-based MOReL and MBOP and return-conditioning DT, but worse than the best offline techniques designed specifically for single-task performance. We also investigated a variant using Diffuser as a dynamics model in conventional trajectory optimizers such as MPPI (Williams et al., 2015), but found

Dataset	Environment	BC	CQL	IQL	AWAC	DT	TT	MOReL	MBOP	Diffuser
Medium-Expert	HalfCheetah	55.2	91.6	86.7	42.8	86.8	95.0	53.3	**105.9**	79.8 ±1.7
Medium-Expert	Hopper	52.5	**105.4**	91.5	55.8	**107.6**	**110.0**	**108.7**	55.1	**107.2** ±1.3
Medium-Expert	Walker2d	**107.5**	**108.8**	**109.6**	74.5	**108.1**	101.9	95.6	70.2	**108.4** ±1.9
Medium	HalfCheetah	42.6	44.0	**47.4**	43.5	42.6	**46.9**	42.1	44.6	44.2 ±0.8
Medium	Hopper	52.9	58.5	66.3	57.0	67.6	61.1	**95.4**	48.8	58.5 ±5.1
Medium	Walker2d	75.3	72.5	**78.3**	72.4	74.0	**79.0**	**77.8**	41.0	**79.7** ±1.5
Medium-Replay	HalfCheetah	36.6	**45.5**	**44.2**	40.5	36.6	41.9	40.2	42.3	42.2 ±2.8
Medium-Replay	Hopper	18.1	**95.0**	**94.7**	37.2	82.7	91.5	**93.6**	12.4	**96.8** ±0.8
Medium-Replay	Walker2d	26.0	77.2	73.9	27.0	66.6	**82.6**	49.8	9.7	61.2 ±6.7
Average		51.9	**77.6**	**77.0**	50.1	74.7	**78.9**	72.9	47.8	**75.3**

Table 2. **(Offline reinforcement learning)** The performance of Diffuser and a variety of prior algorithms on the D4RL locomotion benchmark (Fu et al., 2020). Results for Diffuser correspond to the mean and standard error over 15 random seeds (three independently trained diffusion models and five trajectories per model). We detail the sources for the performance of prior methods in Appendix A.3. Following Kostrikov et al. (2022), we emphasize in bold scores within 5 percent of the maximum per task ($\geq 0.95 \cdot \max$).

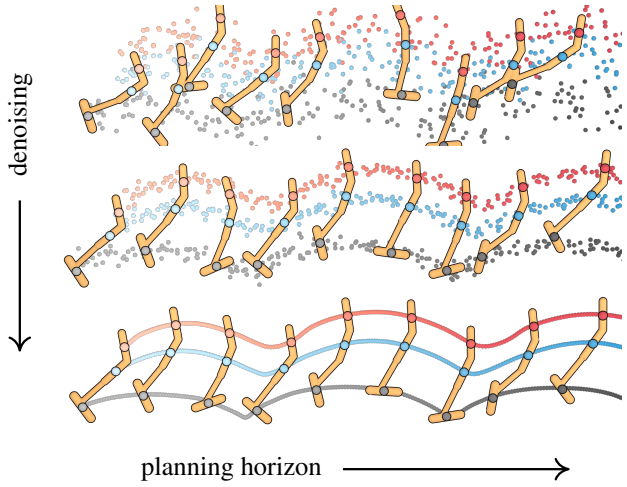

Figure 6. **(Guided sampling)** Diffuser generates all timesteps of a plan concurrently, instead of autoregressively, through the denoising process.

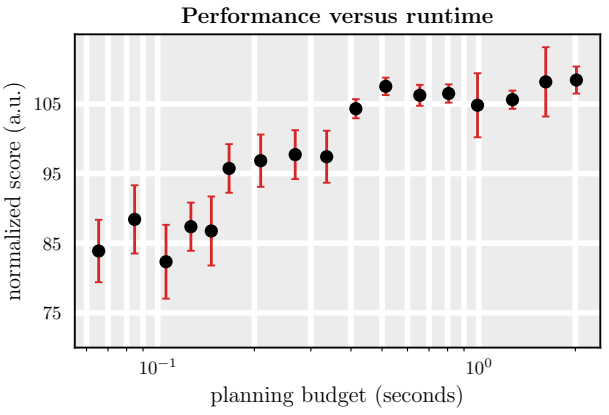

Figure 7. **(Fast Planning)** Performance of Diffuser on Walker2d Medium-Expert when varying the number of diffusion steps to warm-start planning. Performance suffers only minimally even when using one-tenth the number of diffusion steps, as long as plans are initialized from the previous timestep's plan.

that this combination performed no better than random, suggesting that the effectiveness of Diffuser stems from coupled modeling and planning, and not from improved open-loop predictive accuracy.

5.4 Warm-Starting Diffusion for Faster Planning

A limitation of Diffuser is that individual plans are slow to generate (due to iterative generation). Naïvely, as we execute plans open loop, a new plan must be regenerated at each step of execution. To improve execution speed of Diffuser, we may further reuse previously generated plans to warm-start generations of subsequent plans.

To warm-start planning, we may run a limited number of forward diffusion steps from a previously generated plan and then run a corresponding number of denoising steps from this partially noised trajectory to regenerate an updated plan. In Figure 7, we illustrate the trade-off between performance and runtime budget as we vary the underlying number of denoising steps used to regenerate each a new plan from 2 to 100. We find that we may reduce the planning budget of our approach markedly with only modest drop in performance.

6 Related Work

Advances in deep generative modeling have recently made inroads into model-based reinforcement learning, with multiple lines of work exploring dynamics models parameterized as convolutional U-networks (Kaiser et al., 2020), stochastic recurrent networks (Ke et al., 2018; Hafner et al., 2021a; Ha & Schmidhuber, 2018), vector-quantized autoencoders (Hafner et al., 2021b; Ozair et al., 2021), neural ODEs (Du et al., 2020a), normalizing flows (Rhinehart et al., 2020; Janner et al., 2020), generative adversarial networks (Eysenbach et al., 2021), energy-based models (*EBMs*; Du et al. 2019), graph neural networks (Sanchez-Gonzalez et al., 2018), neural radiance fields (Li et al., 2021), and Transformers (Janner et al., 2021; Chen et al., 2021a). These investigations generally assume an abstraction barrier between the model and planner. Specifically, the role of learning is relegated to approximating environment dynamics; once learning is complete the model may be inserted into any of a variety of planning (Botev et al., 2013; Williams et al., 2015) or policy optimization (Sutton, 1990; Wang et al., 2019) algorithms because the form of the planner does not depend strongly on the form of the model. Our goal is to break this abstraction barrier by designing a model and planning algorithm that are trained alongside one another, resulting in a non-autoregressive trajectory-level model for which sampling and planning are nearly identical.

A number of parallel lines of work have studied how to break the abstraction barrier between model learning and planning in different ways. Approaches include training an autoregressive latent-space model for reward prediction (Tamar et al., 2016; Oh et al., 2017; Schrittwieser et al., 2019); weighing model training objectives by state values (Farahmand et al., 2017); and applying collocation techniques to learned single-step energies. In contrast, our method plans by generating all timesteps of a trajectory concurrently, instead of autoregressively, and conditioning the sampled trajectories with auxiliary guidance functions.

Diffusion models have emerged as a promising class of generative model that formulates the data-generating process as an iterative denoising procedure (Sohl-Dickstein et al., 2015; Ho et al., 2020). The denoising procedure

can be seen as parameterizing the gradients of the data distribution (Song & Ermon, 2019), connecting diffusion models to score matching (Hyvärinen, 2005) and EBMs (LeCun et al., 2006; Du & Mordatch, 2019; Nijkamp et al., 2019; Grathwohl et al., 2020). Iterative, gradient-based sampling lends itself towards flexible conditioning (Dhariwal & Nichol, 2021) and compositionality (Du et al., 2020b), which we use to recover effective behaviors from heterogeneous datasets and plan for reward functions unseen during training. While diffusion models have been developed for the generation of images (Song et al., 2021), waveforms (Chen et al., 2021c), 3D shapes (Zhou et al., 2021), and text (Austin et al., 2021), to the best of our knowledge they have not previously been used in the context of reinforcement learning or decision-making.

7 Conclusion

We have presented Diffuser, a denoising diffusion model for trajectory data. Planning with Diffuser is almost identical to sampling from it, differing only in the addition of auxiliary perturbation functions that serve to guide samples. The learned diffusion-based planning procedure has a number of useful properties, including graceful handling of sparse rewards, the ability to plan for new rewards without retraining, and a temporal compositionality that allows it to produce out-of-distribution trajectories by stitching together in-distribution subsequences. Our results point to a new class of diffusion-based planning procedures for deep model-based reinforcement learning.

Code References

We used the following open-source libraries for this work: NumPy (Harris et al., 2020), PyTorch (Paszke et al., 2019), and Diffusion Models in PyTorch (Wang, 2020).

Acknowledgements

We thank Ajay Jain for feedback on an early draft and Leslie Kaelbling, Tomas Lozano-Perez, Jascha Sohl-Dickstein, Ben Eysenbach, Amy Zhang, Colin Li, and Toru Lin for helpful discussions. This work was partially supported by computational resource donations from Microsoft. M.J. is supported by fellowships from the National Science Foundation and the Open Philanthropy Project. Y.D. is supported by a fellowship from the National Science Foundation.

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. Hindsight experience replay. In *Advances in Neural Information Processing Systems*. 2017.

Argenson, A. and Dulac-Arnold, G. Model-based offline planning. In *International Conference on Learning Representations*, 2021.

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and van den Berg, R. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, 2021.

Bapst, V., Sanchez-Gonzalez, A., Doersch, C., Stachenfeld, K. L., Kohli, P., Battaglia, P. W., and Hamrick, J. B. Structured agents for physical construction. In *International Conference on Machine Learning*, 2019.

Botev, Z. I., Kroese, D. P., Rubinstein, R. Y., and L'Ecuyer, P. The cross-entropy method for optimization. In *Handbook of Statistics*, volume 31, chapter 3. 2013.

Chen, C., Yoon, J., Wu, Y.-F., and Ahn, S. TransDreamer: Reinforcement learning with transformer world models, 2021a.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, 2021b.

Chen, N., Zhang, Y., Zen, H., Weiss, R. J., Norouzi, M., and Chan, W. Wavegrad: Estimating gradients for waveform generation. In *International Conference on Learning Representations*, 2021c.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*. 2018.

Dhariwal, P. and Nichol, A. Q. Diffusion models beat GANs on image synthesis. In *Advances in Neural Information Processing Systems*, 2021.

Du, J., Futoma, J., and Doshi-Velez, F. Model-based reinforcement learning for semi-markov decision processes with neural odes. In *Advances in Neural Information Processing Systems*, 2020a.

Du, Y. and Mordatch, I. Implicit generation and generalization in energy-based models. In *Advances in Neural Information Processing Systems*, 2019.

Du, Y., Lin, T., and Mordatch, I. Model based planning with energy based models. In *Conference on Robot Learning*, 2019.

Du, Y., Li, S., and Mordatch, I. Compositional visual generation with energy based models. In *Advances in Neural Information Processing Systems*, 2020b.

Eysenbach, B., Khazatsky, A., Levine, S., and Salakhutdinov, R. Mismatched no more: Joint model-policy optimization for model-based rl. *arXiv preprint arXiv:2110.02758*, 2021.

Farahmand, A.-M., Barreto, A., and Nikovski, D. Value-aware loss function for model-based reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, 2017.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. D4RL: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Fujimoto, S., Meger, D., and Precup, D. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, 2019.

Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *International Conference on Automated Planning and Scheduling*, 2020.

Grathwohl, W., Wang, K.-C., Jacobsen, J.-H., Duvenaud, D., and Zemel, R. Learning the stein discrepancy for training and evaluating energy-based models without sampling. In *International Conference on Machine Learning*, 2020.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, 2018.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, 2021a.

Hafner, D., Lillicrap, T. P., Norouzi, M., and Ba, J. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021b.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020.

Hyvärinen, A. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 2005.

Janner, M., Mordatch, I., and Levine, S. γ-models: Generative temporal difference learning for infinite-horizon prediction. In *Advances in Neural Information Processing Systems*, 2020.

Janner, M., Li, Q., and Levine, S. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2021.

Kaiser, L., Babaeizadeh, M., Miłos, P., Osiński, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G., and Michalewski, H. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2020.

Ke, N. R., Singh, A., Touati, A., Goyal, A., Bengio, Y., Parikh, D., and Batra, D. Modeling the long term future in model-based reinforcement learning. In *International Conference on Learning Representations*, 2018.

Kelly, M. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4): 849–904, 2017.

Kidambi, R., Rajeswaran, A., Netrapalli, P., and Joachims, T. MOReL: Model-based offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Kostrikov, I., Nair, A., and Levine, S. Offline reinforcement learning with implicit Q-learning. In *International Conference on Learning Representations*, 2022.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative Q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

LeCun, Y., Chopra, S., Hadsell, R., Huang, F. J., and et al. A tutorial on energy-based learning. In *Predicting Structured Data*. MIT Press, 2006.

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

Li, Y., Li, S., Sitzmann, V., Agrawal, P., and Torralba, A. 3d neural scene representations for visuomotor control. In *Conference on Robot Learning*, 2021.

Misra, D. Mish: A self regularized non-monotonic neural activation function. In *British Machine Vision Conference*, 2019.

Nagabandi, A., Kahn, G., S. Fearing, R., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *International Conference on Robotics and Automation*, 2018.

Nair, A., Dalal, M., Gupta, A., and Levine, S. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Nichol, A. Q. and Dhariwal, P. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, 2021.

Nijkamp, E., Hill, M., Zhu, S.-C., and Wu, Y. N. Learning non-convergent non-persistent short-run MCMC toward energy-based model. In *Advances in Neural Information Processing Systems*, 2019.

Oh, J., Singh, S., and Lee, H. Value prediction network. In *Advances in Neural Information Processing Systems*, 2017.

Ozair, S., Li, Y., Razavi, A., Antonoglou, I., Van Den Oord, A., and Vinyals, O. Vector quantized models for planning. In *International Conference on Machine Learning*, 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. 2019.

Posa, M., Cantu, C., and Tedrake, R. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 2014.

Rhinehart, N., McAllister, R., and Levine, S. Deep imitative models for flexible inference, planning, and control. In *International Conference on Learning Representations*, 2020.

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, 2018.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, 2015.

Song, J., Meng, C., and Ermon, S. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021.

Song, Y. and Ermon, S. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems*, 2019.

Sutton, R. S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *International Conference on Machine Learning*, 1990.

Talvitie, E. Model regularization for stable sample rollouts. In *Conference on Uncertainty in Artificial Intelligence*, 2014.

Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. Value iteration networks. In *Advances in Neural Information Processing Systems*. 2016.

Tassa, Y., Erez, T., and Todorov, E. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *International Conference on Intelligent Robots and Systems*, 2012.

Wang, P. Implementation of denoising diffusion probabilistic models in pytorch, 2020. URL https://github.com/lucidrains/denoising-diffusion-pytorch.

Wang, T., Bao, X., Clavera, I., Hoang, J., Wen, Y., Langlois, E., Zhang, S., Zhang, G., Abbeel, P., and Ba, J. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

Williams, G., Aldrich, A., and Theodorou, E. Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*, 2015.

Witkin, A. and Kass, M. Spacetime constraints. *ACM Siggraph Computer Graphics*, 1988.

Wu, Y. and He, K. Group normalization. In *European Conference on Computer Vision*, 2018.

Zhou, L., Du, Y., and Wu, J. 3D shape generation and completion through point-voxel diffusion. In *International Conference on Computer Vision*, 2021.

Appendix A Baseline details and sources

In this section, we provide details about baselines we ran ourselves. For scores of baselines previously evaluated on standardized tasks, we provide the source of the listed score.

A.1 Maze2D experiments

Single-task. The performance of CQL and IQL on the standard Maze2D environments is reported in the D4RL whitepaper (Fu et al., 2020) in Table 2.

We ran IQL using the offical implementation from the authors:

`github.com/ikostrikov/implicit_q_learning`.

We tuned over two hyperparameters:

1. temperature $\in [3, 10]$
2. expectile $\in [0.65, 0.95]$

Multi-task. We only evaluated IQL on the Multi2D environments because it is the strongest baseline in the single-task Maze2D environments by a sizeable margin. To adapt IQL to the multi-task setting, we modified the Q-functions, value function, and policy to be goal-conditioned. To select goals during training, we employed a strategy based on hindsight experience replay, in which we sampled a goal from among those states encountered in the future of a trajectory. For a training backup $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$, we sampled goals according to a geometric distribution over the future

$$\Delta \sim \text{Geom}(1-\gamma) \qquad \mathbf{g} = \mathbf{s}_{t+\Delta},$$

recalculated rewards based on the sampled goal, and conditioned all relevant models on the goal during updating. During testing, we conditioned the policy on the ground-truth goal.

We tuned over the same IQL parameters as in the single-task setting.

A.2 Block stacking experiments

Single-task. We ran CQL using the following implementation

`https://github.com/young-geng/cql`.

and used default hyperparameters in the code. We ran BCQ using the author's original implementation

`https://github.com/sfujim/BCQ`.

For BCQ, we tuned over two hyperparameters:

1. discount factor $\in [0.9, 0.999]$
2. tau $\in [0.001, 0.01]$

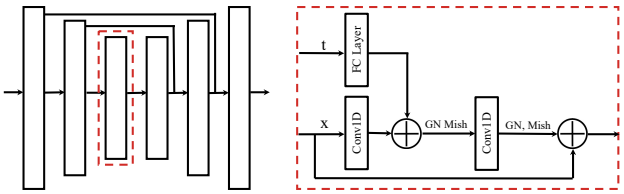

Figure A1. Diffuser has a U-Net architecture with residual blocks consisting of temporal convolutions, group normalization, and Mish nonlinearities.

Multi-task. To evaluate BCQ and CQL in the multi-task setting, we modified the Q-functions, value function and policy to be goal-conditioned. We trained using goal relabeling as in the Multi2D environments. We tuned over the same hyperparameters described in the single-task block stacking experiments.

A.3 Offline Locomotion

The scores for BC, CQL, IQL, and AWAC are from Table 1 in Kostrikov et al. (2022). The scores for DT are from Table 2 in Chen et al. (2021b). The scores for TT are from Table 1 in Janner et al. (2021). The scores for MOReL are from Table 2 in Kidambi et al. (2020). The scores for MBOP are from Table 1 in Argenson & Dulac-Arnold (2021).

Appendix B Test-time Flexibility

To guide Diffuser to stack blocks in specified configurations, we used two separate perturbation functions $h(\boldsymbol{\tau})$ to specify a given stack of block A on top of block B, which we detail below.

Final State Matching To enforce a final state consisting of block A on top of block B, we trained a perturbation function $h_{\text{match}}(\boldsymbol{\tau})$ as a per-timestep classifier determining whether a a state $\boldsymbol{s}$ exhibits a stack of block A on top of block B. We train the classifier on the demonstration data as the diffusion model.

Contact Constraint To guide the Kuka arm to stack block A on top of block B, we construct a perturbation function $h_{\text{contact}}(\boldsymbol{\tau}) = \sum_{i=0}^{64} -1 * \|\boldsymbol{\tau}_{c_i} - 1\|^2$, where $\boldsymbol{\tau}_{c_i}$ corresponds to the underlying dimension in state $\boldsymbol{\tau}_{s_i}$ that specifies the presence or absence of contact between the Kuka arm and block A. We apply the contact constraint between the Kuka arm and block A for the first 64 timesteps in a trajectory, corresponding to initial contact with block A in a plan.

Appendix C Implementation Details

In this section we describe the architecture and record hyperparameters.

1. The architecture of Diffuser (Figure A1) consists of a U-Net structure with 6 repeated residual blocks. Each block consisted of two temporal convolutions, each followed by group norm (Wu & He, 2018), and a final Mish nonlinearity (Misra, 2019). Timestep embeddings are produced by a single fully-connected layer and added to the activations of the first temporal convolution within each block.

2. We train the model using the Adam optimizer (Kingma & Ba, 2015) with a learning rate of $4\text{e}{-05}$ and batch size of 32. We train the models for 500k steps.

3. The return predictor $\mathcal{J}$ has the structure of the first half of the U-Net used for the diffusion model, with a final linear layer to produce a scalar output.

4. We use a planning horizon T of 100 in all locomotion tasks, 128 for blocking stacking, 128 in Maze2D / Multi2D U-Maze, 265 in Maze2D / Multi2D Medium, and 384 in Maze2D / Multi2D Large.

5. We use $N = 100$ diffusion steps.

6. We use a guide scale of $\alpha = 0.001$.

7. We tuned over discount factors for the return prediction $\mathcal{J}_\phi$, but found that above $\gamma = 0.99$ planning was insensitive to changes in discount factor.

8. We found that control performance was not substantially affected by the choice of predicting noise ϵ versus uncorrupted data $\boldsymbol{\tau}^0$ with the diffusion model.