Title
Deep Generative Models for Decision-Making and Control

Permalink
https://escholarship.org/uc/item/7b39125c

Author
Janner, Michael

Publication Date
2023

Peer reviewed|Thesis/dissertation

Deep Generative Models for Decision-Making and Control

By

Michael Janner

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sergey Levine, Chair
Professor Anca Dragan
Professor Jacob Steinhardt
Professor Karthik Narasimhan

Spring 2023

Deep Generative Models for Decision-Making and Control

Copyright 2023

by

Michael Janner

Abstract

Deep model-based reinforcement learning methods offer a conceptually simple approach to the decision-making and control problem: use learning for the purpose of estimating an approximate dynamics model, and offload the rest of the work to classical trajectory optimization. However, this combination has a number of empirical shortcomings, limiting the usefulness of model-based methods in practice. The dual purpose of this thesis is to study the reasons for these shortcomings and to propose solutions for the uncovered problems. We begin by generalizing the dynamics model itself, replacing the standard single-step formulation with a model that predicts over probabilistic latent horizons. The resulting model, trained with a generative reinterpretation of temporal difference learning, leads to infinite-horizon variants of the procedures central to model-based control, including the model rollout and model-based value estimation.

Next, we show that poor predictive accuracy of commonly-used deep dynamics models is a major bottleneck to effective planning, and describe how to use high-capacity sequence models to overcome this limitation. Framing reinforcement learning as sequence modeling simplifies a range of design decisions, allowing us to dispense with many of the components normally integral to reinforcement learning algorithms. However, despite their predictive accuracy, such sequence models are limited by the search algorithms in which they are embedded. As such, we demonstrate how to fold the entire trajectory optimization pipeline into the generative model itself, such that sampling from the model and planning with it become nearly identical. The culmination of this endeavor is a method that improves its planning capabilities, and not just its predictive accuracy, with more data and experience. Along the way, we highlight how inference techniques from the contemporary generative modeling toolbox, including beam search, classifier-guided sampling, and image inpainting, can be reinterpreted as viable planning strategies for reinforcement learning problems.

Contents

List of Figures

List of Tables

Acknowledgments

I am luckier than I deserve in having so many people to thank.

First and foremost, I thank my advisor, Sergey Levine, for sharpening my thinking and helping me develop a research taste during the past five years. Sergey has a remarkable ability to always find time to chat about research regardless of how busy he is, and my work benefitted immensely from those many conversations. I am also grateful to my committee, Anca Dragan, Jacob Steinhardt, and Karthik Narasimhan, for their feedback and perspectives that have improved this dissertation.

I decided to pursue a PhD in artificial intelligence due to an overwhelmingly positive research experience as an undergrad at MIT. Working with Josh Tenenbaum, Regina Barzilay, and Bill Freeman as well as their (at the time) students and postdocs Tejas Kulkarni, Karthik Narasimhan, Jiajun Wu, Ilker Yildirim, Pedro Tsividis, and Max Kleiman-Weiner was the best introduction to the field imaginable.

Before starting at MIT, I spent a few formative years in Yadong Yin's materials science group at UC Riverside. Here I met my first research mentors: Qiao Zhang, Le He, and Mingsheng Wang. Looking back, I only grow more impressed at their patience and generosity. I am sure they had much more pressing things to attend to than showing a high school student the research ropes, but they did so anyway. It is because of Yadong and his students that I am a scientist today.

Over the course of my PhD, I have been fortunate to work with many collaborators. Igor Mordatch, Colin Li, Yilun Du, Kevin Black, Chelsea Finn, Justin Fu, JD Co-Reyes, Rishi Veerapaneni, Katie Kang, Ilya Kostrikov, Philippe Hansen-Estruch, Zhengyao Jiang, Tianjun Zhang, Yueying Li, and Yuandong Tian have all taught me so much and expanded my research horizons.

The Berkeley AI Research lab was a wonderful place to study. Aviral Kumar, Young Geng, Dibya Ghosh, Laura Smith, Philip Ball, Manan Tomar, Oleh Rybkin, Marwa Abdulhai, Kuba Grudzien, Dhruv Shah, Charlie Snell, Homer Walke, Simon Zhai, Coline Devin, Abhishek Gupta, Anusha Nagabandi, Natasha Jaques, Dinesh Jarayaman, Rowan McAllister, Vitchyr Pong, Kelvin Xu, Amy Zhang, Glen Berseth, Aurick Zhou, Avi Singh, Ashvin Nair, Allan Jabri, Sasha Sax, Vickie Ye, and the rest of the BAIR community made me look forward to coming into the lab every day. I am especially grateful to Michael Chang, for being my sounding board on nearly everything, and Marvin Zhang, for keeping me sane during a pandemic.

During the summer before my final year at Berkeley, I took a research detour and worked on reinforcement learning for program synthesis applications at Google. I would like to thank my host Alex Polozov, for a gracious introduction to a new subfield, as well as my residency

1

Introduction

This thesis examines one of the simplest conceivable strategies for data-driven decision-making and robotic control problems. The abstract procedure consists of two interleaved steps:

1. Use data to fit a parametric model used to predict the future given the past.

2. Use the model to predict the outcomes of a candidate set of action sequences, selecting that which produces the most desirable result.

This high-level description outlines a type of model-predictive control algorithm that uses "planning in the now" (Kaelbling & Lozano-Pérez, 2011; van Hasselt et al., 2019), meaning that the model is used to predict into the future while making decisions as opposed to other ways of using model-generated data. It leaves much to be specified: how does one choose the candidate actions for evaluation? How should the model be structured? What constitutes useful data?

Regardless, this specification is already sufficient to suggest why it might be a good approach. Step 1 amounts to supervised learning, which now often works reliably given enough data and high-capacity function approximators like neural networks (Krizhevsky et al., 2012; Zhang et al., 2017; Kaplan et al., 2020). In control contexts, step 2 can (in principle) be offloaded to trajectory optimization algorithms, which have been the subject of much study and are similarly well-understood in their original context when the ground-truth dynamics are known (Diehl et al., 2009; Tassa et al., 2012; Kelly, 2017). It would appear that this approach combines two fairly reliable puzzle pieces.

Moreover, the separation between model-learning and decision-making has a number of appealing properties. Most obviously, it allows for reuse of the learned model, allowing it to be deployed for a variety of tasks in the same setting. This level of reuse is not as straightforward with model-free approaches because the reward function cannot be separated

from the implicit dynamics knowledge encoded in a learned policy or value function. This property also allows for a model to be trained from data that is not explicitly labeled with rewards, which can be useful in situations where rewards are difficult to define but experience is plentiful. Empirically, dynamics models are found to be easier to train than value functions, allowing for better sample efficiency and generalization of learned models (Janner et al., 2019); this can be viewed as a consequence of either differences between the types of algorithms used to train value functions versus dynamics models (Kumar et al., 2022) or the simplicity of the dynamics itself relative to the optimal value function (Dong et al., 2020). Finally, this separation provides a convenient way to interpret the learned model: for any decision a planning routine produces, one can inspect the model-expected outcome that caused that decision to be selected.

Unfortunately, employing this strategy is not as straightforward as it might seem, nor do these purported benefits always translate to practice. While there have been successful demonstrations of the combination (Chua et al., 2018; Argenson & Dulac-Arnold, 2021), it is surprisingly difficult to extract a set of design principles from these successes that allow for the approach to be effectively applied to new problems without extensive problem-specific tuning. As a result, the contemporary frontier of deep model-based reinforcement learning consists largely of algorithms that pull extensively from the model-*free* reinforcement learning toolbox. By contrast, conventional planning in the now with deep neural networks is rare.

This state of affairs should be surprising. The dual purpose of the following chapters is to explain why this is the case and to suggest a way forward. After a brief description of the problem setting and review of technical background in Chapter 2, we proceed to three primary ideas:

- In Chapter 3, we reconsider the role of state-space prediction in reinforcement learning. The result is a model that amortizes the work of prediction during training time, much like a value function, as opposed to relying on model-based rollouts. As a result, the model can predict over infinite probabilistic horizons without sequential rollouts, blurring the line between model-based and model-free mechanisms. This investigation underscores a particular drawback: representing high-dimensional joint distributions over future trajectories is a difficult generative modeling problem.

- In Chapter 4, we ask whether the quality of the predictive model is the bottleneck. We appeal to recent successes in generative modeling and replace the conventional single-step dynamics model with a long-horizon Transformer. In the process, we show how to reinterpret algorithms from the sequence modeling toolbox as viable planning algorithms.

- Transformers largely address the predictive quality bottleneck, but are still limited by the quality of the planning routine in which they are embedded. In Chapter 5, we discuss a way of incorporating both the prediction and the planning into a generative

model, such that the line between sampling from the model and planning with it becomes blurred. The end result is a method that improves its planning capabilities, and not just its predictive accuracy, with more data and experience.

We conclude by discussing the lessons learned from these investigations and their implications for future model-based reinforcement learning algorithms in Chapter 6.

2

Preliminaries

This brief chapter introduces the problem setting studied by this thesis and defines notation.

The reinforcement learning problem. We consider an infinite-horizon Markov decision process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma, \rho_0)$, with state space $\mathcal{S}$ and action space $\mathcal{A}$. The transition distribution and reward function are given by $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^+$ and $r : \mathcal{S} \to \mathbb{R}$, respectively. The discount is denoted by $\gamma \in [0, 1)$ and the initial state distribution by $\rho_0 : \mathcal{S} \to \mathbb{R}^+$. A policy $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+$ describes the distribution over actions taken at a particular state. The goal of the reinforcement learning problem is to find the optimal policy π^* that maximizes the expected sum of discounted rewards:

$$\pi^* = \arg\max_{\pi} \ \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \right] . \tag{2.1}$$

The discounted occupancy. A policy π induces a conditional occupancy $\mu^{\pi}(\mathbf{s} \mid \mathbf{s}_t, \mathbf{a}_t)$ over future states:

$$\mu^{\pi}(\mathbf{s} \mid \mathbf{s}_t, \mathbf{a}_t) = (1 - \gamma) \sum_{\Delta t=1}^{\infty} \gamma^{\Delta t - 1} p(\mathbf{s}_{t+\Delta t} = \mathbf{s} \mid \mathbf{s}_t, \mathbf{a}_t, \pi). \tag{2.2}$$

The discounted occupancy is a distribution over states encountered by the policy when using a geometric weighting over future timesteps, analogous to the geometric weighting of the reinforcement learning objective in Equation 2.1. Unlike the single-step transition distribution p, the discounted occupancy is policy-conditioned because it marginalizes over

4

future action distributions. When $\gamma = 0$, the discounted occupancy μ^π becomes policy-agnostic and identical to the single-step transition distribution p. For brevity, we omit the policy superscript π in the discounted occupancy μ^π when it is otherwise clear from context.

The optimization objective Equation 2.1 can be reformulated as the expected reward over the policy-conditioned discounted occupancy:

$$\mathbb{E}_\pi \left[\sum_{t=0}^\infty \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \right] = \mathbb{E}_{\substack{\mathbf{s}_0 \sim \rho_0(\cdot) \\ \mathbf{s}_t \sim \mu(\cdot|\mathbf{s}_0, \mathbf{a}_0) \\ \mathbf{a}_t \sim \pi(\cdot|\mathbf{s}_t)}} [r(\mathbf{s}_t, \mathbf{a}_t)] \tag{2.3}$$

Function approximation. In the reinforcement learning problem, we assume only the ability to interact in the environment, which provides data streams in the form of trajectories. We do not assume query access to the functional form of, for example, the transition distribution p or reward function r. Instead, if these are used by an algorithm, they must be approximated from data. We denote parametric approximations of p (or μ) as p_θ (or μ_θ), in which the subscripts denote model parameters.

Trajectory optimization. Trajectory optimization (Witkin & Kass, 1988; Tassa et al., 2012) 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) $r(\mathbf{s}_t, \mathbf{a}_t)$.

$$\mathbf{a}_{0:T}^* = \arg\max_{\mathbf{a}_{0:T}} \mathcal{J}(\mathbf{s}_0, \mathbf{a}_{0:T}) = \arg\max_{\mathbf{a}_{0:T}} \sum_{t=0}^T r(\mathbf{s}_t, \mathbf{a}_t) \tag{2.4}$$

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.

This problem statement is similar to that in Equation 2.1, and it is commonplace to use trajectory optimization algorithms to address problems formulated in the language of reinforcement learning. However, there are two differences worth discussing. The most apparent distinction is that the trajectory optimization objective in Equation 2.4 considers a finite-horizon decision-making problem, though in practice the use of terminal value functions trained via reinforcement learning can allow for the consideration of infinite horizons as well. The second distinction is that the optimization variables are of a different data type: instead of functions that output actions, they are now primitive actions themselves. Optimizing over actions directly allows for the representation of non-Markovian policies. For example, the solution to Equation 2.4 could prescribe a different action to be taken every time a particular state is encountered; the solution to Equation 2.1 could not without changing the definition of a policy.

3

Infinite-Horizon Prediction

3.1 Introduction

The common ingredient in all of model-based reinforcement learning is the dynamics model: a function used for predicting future states. The choice of the model's prediction horizon constitutes a delicate trade-off. Shorter horizons make the prediction problem easier, as the near-term future increasingly begins to look like the present, but may not provide sufficient information for decision-making. Longer horizons carry more information, but present a more difficult prediction problem, as errors accumulate rapidly when a model is applied to its own previous outputs (Talvitie, 2014).

Can we avoid choosing a prediction horizon altogether? Value functions already do so by modeling the cumulative return over a discounted long-term future instead of an immediate reward, circumventing the need to commit to any single finite horizon. However, value prediction folds two problems into one by entangling environment dynamics with reward structure, making value functions less readily adaptable to new tasks in known settings than their model-based counterparts.

In this chapter, we propose a model that predicts over an infinite horizon with a geometrically-distributed timestep weighting (Figure 5.1). This γ-model, named for the dependence of its probabilistic horizon on a discount factor γ, is trained with a generative analogue of temporal difference learning suitable for continuous spaces. The γ-model bridges the gap between canonical model-based and model-free mechanisms. Like a value function, it is policy-conditioned and contains information about the distant future; like a conventional dynamics model, it is independent of reward and may be reused for new tasks within the same environment. The γ-model may be instantiated as both a generative adversarial network (Goodfellow et al., 2014a) and a normalizing flow (Rezende & Mohamed, 2015).

single-step model: $\Delta t = 1$ **γ-model**: $\Delta t \sim \mathrm{Geom}(1 - \gamma)$

current state prediction
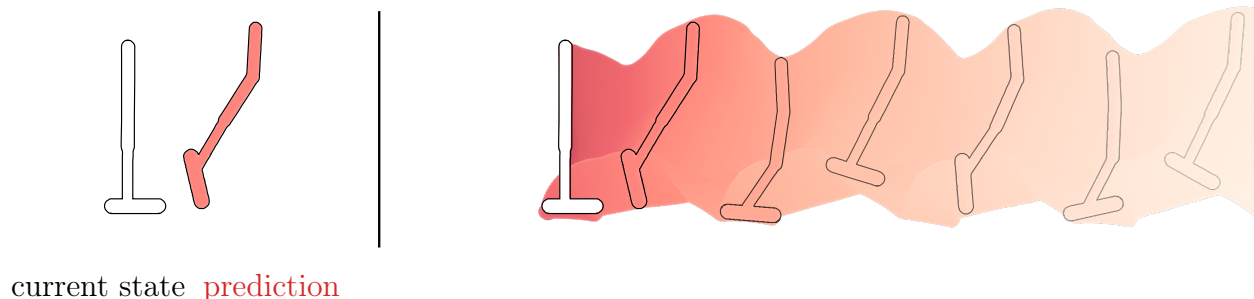
Figure 3.1: **(Prediction with probabilistic horizons)** Conventional predictive models trained via maximum likelihood have a horizon of one. By interpreting temporal difference learning as a training algorithm for generative models, it is possible to predict with a probabilistic horizon governed by a geometric distribution. In the spirit of infinite-horizon control in model-free reinforcement learning, we refer to this formulation as infinite-horizon prediction.

The shift from standard single-step models to infinite-horizon γ-models carries several advantages:

Constant-time prediction   Single-step models must perform an $\mathcal{O}(n)$ operation to predict n steps into the future; γ-models amortize the work of predicting over extended horizons during training such that long-horizon prediction occurs with a single feedforward pass of the model.

Generalized rollouts and value estimation   Probabilistic prediction horizons lead to generalizations of the core procedures of model-based reinforcement learning. For example, generalized rollouts allow for fine-grained interpolation between training-time and testing-time compounding error. Similarly, terminal value functions appended to truncated γ-model rollouts allow for a gradual transition between model-based and model-free value estimation.

Omission of unnecessary information   The predictions of a γ-model do not come paired with an associated timestep. While on the surface a limitation, we show why knowing precisely *when* a state will be encountered is not necessary for decision-making. Infinite-horizon γ-model prediction selectively discards the unnecessary information from a standard model-based rollout.

3.2 Related Work

The complementary strengths and weaknesses of model-based and model-free reinforcement learning have led to a number of works that attempt to combine these approaches. Common strategies include initializing a model-free algorithm with the solution found by a model-based planner (Levine & Koltun, 2013; Farshidian et al., 2014; Nagabandi et al., 2018), feeding model-generated data into an otherwise model-free optimizer (Sutton, 1990; Silver et al., 2008; Lampe & Riedmiller, 2014; Kalweit & Boedecker, 2017; Luo et al., 2019), using model predictions to improve the quality of target values for temporal difference learning (Buckman et al., 2018b; Feinberg et al., 2018), leveraging model gradients for backpropagation (Nguyen & Widrow, 1990; Jordan & Rumelhart, 1992; Heess et al., 2015b), and incorporating model-based planning without explicitly predicting future observations (Tamar et al., 2016; Silver et al., 2017; Oh et al., 2017; Kahn et al., 2018; Amos et al., 2018; Schrittwieser et al., 2019). In contrast to combining independent model-free and model-based components, we describe a framework for training a new class of predictive model with a generative, model-based reinterpretation of model-free tools.

Temporal difference models (TDMs) (Pong et al., 2018) provide an alternative method of training models with what are normally considered to be model-free algorithms. TDMs interpret models as a special case of goal-conditioned value functions (Kaelbling, 1993; Foster & Dayan, 2002; Schaul et al., 2015; Andrychowicz et al., 2017), though the TDM is constrained to predict at a fixed horizon and is limited to tasks for which the reward depends only on the last state. In contrast, the γ-model predicts over a discounted infinite-horizon future and accommodates arbitrary rewards.

The most closely related prior work to γ-models is the successor representation (Dayan, 1993), a formulation of long-horizon prediction that has been influential in both cognitive science (Momennejad et al., 2017; Gershman, 2018) and machine learning (Kulkarni et al., 2016; Ma et al., 2018). In its original form, the successor representation is tractable only in tabular domains. Prior continuous variants have focused on policy evaluation based on expected state featurizations (Barreto et al., 2017; 2018; Hansen et al., 2020), forgoing an interpretation as a probabilistic model suitable for state prediction. Converting the tabular successor representation into a continuous generative model is non-trivial because the successor representation implicitly assumes the ability to normalize over a finite state space for interpretation as a predictive model.

Because of the discounted state occupancy's central role in reinforcement learning, its approximation by Bellman equations has been the focus of multiple lines of work. Generalizations include β-models (Sutton, 1995), allowing for arbitrary mixture distributions over time, and option models (Sutton et al., 1999), allowing for state-dependent termination conditions. While our focus is on generative models featuring the state-independent geometric timestep weighting of the successor representation, we are hopeful that the tools developed

in this paper could also be applicable in the design of continuous analogues of these generalizations.

3.3 Generative Temporal Difference Learning

Our goal is to make long-horizon predictions without the need to repeatedly apply a single-step model. Instead of modeling states at a particular instant in time by approximating the environment transition distribution $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$, we aim to predict a weighted distribution over all possible future states according to $\mu(\mathbf{s} \mid \mathbf{s}_t, \mathbf{a}_t)$. In principle, this can be posed as a conventional maximum likelihood problem:

$$\max_{\theta} \ \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t, \mathbf{s} \sim \mu(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)} \left[\log \mu_{\theta}(\mathbf{s} \mid \mathbf{s}_t, \mathbf{a}_t) \right].$$

However, doing so would require collecting samples from the occupancy μ independently for each policy of interest. Forgoing the ability to re-use data from multiple policies when training dynamics models would sacrifice the sample efficiency that often makes model usage compelling in the first place, so we instead aim to design an off-policy algorithm for training μ_{θ}. We accomplish this by reinterpreting temporal difference learning as a method for training generative models.

Instead of collecting only on-policy samples from $\mu(\mathbf{s} \mid \mathbf{s}_t, \mathbf{a}_t)$, we observe that μ admits a convenient recursive form. Consider a modified MDP in which there is a $1 - \gamma$ probability of terminating at each timestep. The distribution over the state at termination, denoted as the exit state $\mathbf{s}_e$, corresponds to first sampling from a termination timestep $\Delta t \sim \mathrm{Geom}(1 - \gamma)$ and then sampling from the per-timestep distribution $p(\mathbf{s}_{t+\Delta t} \mid \mathbf{s}_t, \mathbf{a}_t, \pi)$. The distribution over $\mathbf{s}_e$ corresponds exactly to that in the definition of the occupancy μ in Equation 2.2, but also lends itself to an interpretation as a mixture over only two components: the distribution at the immediate next timestep, in the event of termination, and that over all subsequent timesteps, in the event of non-termination. This mixture yields the following target distribution:

$$p_{\mathrm{targ}}(\mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t) = \underbrace{(1 - \gamma)p(\mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t)}_{\text{single-step distribution}} + \underbrace{\gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim p(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)} \left[\mu_{\theta}(\mathbf{s}_e \mid \mathbf{s}_{t+1}) \right]}_{\text{model bootstrap}}. \tag{3.1}$$

We use the shorthand $\mu_{\theta}(\mathbf{s}_e \mid \mathbf{s}_{t+1}) = \mathbb{E}_{\mathbf{a}_{t+1} \sim \pi(\cdot \mid \mathbf{s}_{t+1})} \left[\mu_{\theta}(\mathbf{s}_e \mid \mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \right]$. The target distribution p_{targ} is reminiscent of a temporal difference target value: the state-action conditioned occupancy $\mu_{\theta}(\mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t)$ acts as a Q-function, the state-conditioned occupancy $\mu_{\theta}(\mathbf{s}_e \mid \mathbf{s}_{t+1})$ acts as a value function, and the single-step distribution $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ acts as a reward function. However, instead of representing a scalar target value, p_{targ} is a distribution from which we may sample future states $\mathbf{s}_e$. We can use this target distribution in place of samples

from the true discounted occupancy μ:

$$\max_{\theta} \ \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_e \sim (1-\gamma)p(\cdot|\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}[\mu_\theta(\cdot|\mathbf{s}_{t+1})]} \left[\log \mu_\theta(\mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t) \right].$$

This formulation differs from a standard maximum likelihood learning problem in that the target distribution depends on the current model. By bootstrapping the target distribution in this manner, we are able to use only empirical $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ transitions from one policy in order to train an infinite-horizon predictive model μ_θ for any other policy. Because the horizon is governed by the discount γ, we refer to such a model as a γ-model.

This bootstrapped model training may be incorporated into a number of different generative modeling frameworks. We discuss two cases here. (1) When the model μ_θ permits only sampling, we may train μ_θ by minimizing an f-divergence from samples:

$$\mathcal{L}_1(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) = D_f(\mu_\theta(\cdot \mid \mathbf{s}_t, \mathbf{a}_t) \mid\mid (1-\gamma)p(\cdot \mid \mathbf{s}_t, \mathbf{a}_t) + \gamma\mu_\theta(\cdot \mid \mathbf{s}_{t+1})). \tag{3.2}$$

This objective leads naturally to an adversarially-trained γ-model. (2) When the model μ_θ permits density evaluation, we may minimize an error defined on log-densities directly:

$$\mathcal{L}_2(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) = \mathbb{E}_{\mathbf{s}_e}\left[\left\| \log \mu_\theta(\mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t) - \log\left((1-\gamma)p(\mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t) + \gamma\mu_\theta(\mathbf{s}_e \mid \mathbf{s}_{t+1})\right)\right\|_2^2\right]. \tag{3.3}$$

This objective is suitable for γ-models instantiated as normalizing flows. Due to the approximation of a target log-density $\log\left((1-\gamma)p(\cdot \mid \mathbf{s}_t, \mathbf{a}_t) + \gamma\mathbb{E}_{\mathbf{s}_{t+1}}\left[\mu_\theta(\cdot \mid \mathbf{s}_{t+1})\right]\right)$ using a single next state $\mathbf{s}_{t+1}$, $\mathcal{L}_2$ is unbiased for deterministic dynamics and a bound in the case of stochastic dynamics. We provide complete algorithmic descriptions of both variants and highlight practical training considerations in Section 3.5.

3.4   Analysis and Applications of γ-Models

Using the γ-model for prediction and control requires us to generalize procedures common in model-based reinforcement learning. In this section, we derive the γ-model rollout and show how it can be incorporated into a reinforcement learning procedure that hybridizes model-based and model-free value estimation. First, however, we show that the γ-model is a continuous, generative counterpart to another type of long-horizon model: the successor representation.

γ-Models as a Continuous Successor Representation

The successor representation M is a prediction of expected visitation counts (Dayan, 1993). It has a recurrence relation making it amenable to tabular temporal difference algorithms:

$$M(\mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\mathbf{s}_{t+1}\sim p(\cdot|\mathbf{s}_t, \mathbf{a}_t)}\left[\mathbb{1}\left[\mathbf{s}_e = \mathbf{s}_{t+1}\right] + \gamma M(\mathbf{s}_e \mid \mathbf{s}_{t+1})\right]. \tag{3.4}$$

Adapting the successor representation to continuous state spaces in a way that retains an interpretation as a probabilistic model has proven challenging. However, variants that forego the ability to sample in favor of estimating expected state features have been developed (Barreto et al., 2017).

The form of the successor recurrence relation bears a striking resemblance to that of the target distribution in Equation 3.1, suggesting a connection between the generative, continuous γ-model and the discriminative, tabular successor representation. We now make this connection precise.

Proposition 1. *The global minimum of both $\mathcal{L}_1$ and $\mathcal{L}_2$ is achieved if and only if the resulting γ-model produces samples according to the normalized successor representation:*

$$\mu_\theta(\mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t) = (1 - \gamma) M(\mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t).$$

Proof. In the case of either objective, the global minimum is achieved only when

$$\mu_\theta(\mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t) = (1 - \gamma) p(\mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim p(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)} \left[\mu_\theta(\mathbf{s}_e \mid \mathbf{s}_{t+1}) \right]$$

for all $\mathbf{s}_t, \mathbf{a}_t$. We recognize this optimality condition exactly as the recurrence defining the successor representation M (Equation 3.4), scaled by $(1 - \gamma)$ such that μ_θ integrates to 1 over $\mathbf{s}_e$. $\square$

γ-Model Rollouts

Standard single-step models, which correspond to γ-models with $\gamma = 0$, can predict multiple steps into the future by making iterated autoregressive predictions, conditioning each step on their own output from the previous step. These sequential rollouts form the foundation of most model-based reinforcement learning algorithms. We now generalize these rollouts to γ-models for $\gamma > 0$, allowing us to decouple the discount used during model training from the desired horizon in control. When working with multiple discount factors, we explicitly condition an occupancy on its discount as $\mu(\mathbf{s}_e \mid \mathbf{s}_t; \gamma)$. In the results below, we omit the model parameterization θ whenever a statement applies to both a discounted occupancy μ and a parametric γ-model μ_θ.

Theorem 1. *Let $\mu_n(\mathbf{s}_e \mid \mathbf{s}_t; \gamma)$ denote the distribution over states at the nth sequential step of a γ-model rollout beginning from state $\mathbf{s}_t$. For any desired discount $\tilde{\gamma} \in [\gamma, 1)$, we may reweight the samples from these model rollouts according to the weights*

$$\alpha_n = \frac{(1 - \tilde{\gamma})(\tilde{\gamma} - \gamma)^{n-1}}{(1 - \gamma)^n}$$

to obtain the state distribution drawn from $\mu_1(\mathbf{s}_e \mid \mathbf{s}_t; \tilde{\gamma}) = \mu(\mathbf{s}_e \mid \mathbf{s}_t; \tilde{\gamma})$. That is, we may reweight the steps of a γ-model rollout so as to match the distribution of a $\tilde{\gamma}$-model with larger discount:

$$\mu(\mathbf{s}_e \mid \mathbf{s}_t; \tilde{\gamma}) = \sum_{n=1}^{\infty} \alpha_n \mu_n(\mathbf{s}_e \mid \mathbf{s}_t; \gamma).$$

Proof. Each step of the γ-model samples a time according to $\Delta t \sim \text{Geom}(1 - \gamma)$, so the time after n γ-model steps is distributed according to the sum of n independent geometric random variables with identical parameters. This sum corresponds to a negative binomial random variable, $\text{NB}(n, 1 - \gamma)$, with the following pmf:

$$p_n(t) = \binom{t-1}{t-n} \gamma^{(t-n)} (1-\gamma)^n. \tag{3.5}$$

Equation 3.5 is mildly different from the textbook pmf because we want a distribution over the total number of trials (in our case, cumulative timesteps t) instead of the number of successes before the nth failure. The latter is more commonly used because it gives the random variable the same support, $t \geq 0$, for all n. The form in Equation 3.5 only has support for $t \geq n$, which substantially simplifies the following analysis.

The distributions $q(t)$ expressible as a mixture over the per-timestep negative binomial distributions p_n are given by:

$$q(t) = \sum_{n=1}^{t} \alpha_n p_n(t),$$

in which α_n are the mixture weights. Because p_n only has support for $t \geq n$, it suffices to only consider the first t γ-model steps when solving for $q(t)$.

We are interested in the scenario in which $q(t)$ is also a geometric random variable with smaller parameter, corresponding to a larger discount $\tilde{\gamma}$. We proceed by setting $q(t) = \text{Geom}(1 - \tilde{\gamma})$ and solving for the mixture weights α_n by induction.

Base case. Let $n = 1$. Because p_1 is the only mixture component with support at $t = 1$, α_1 is determined by $q(1)$:

$$1 - \tilde{\gamma} = \alpha_1 \binom{t-1}{t-1} \gamma^{t-1} (1-\gamma)^t$$
$$= \alpha_1 (1 - \gamma).$$

Solving for α_1 gives:

$$\alpha_1 = \frac{1 - \tilde{\gamma}}{1 - \gamma}.$$

Induction step. We now assume the form of α_k for $k = 1, \ldots, n-1$ and solve for α_n using $q(n)$.

$$
\begin{aligned}
(1 - \tilde{\gamma})\tilde{\gamma}^{n-1} &= \sum_{k=1}^{n} \alpha_k \binom{n-1}{n-k} \gamma^{n-k}(1 - \gamma)^k \\
&= \left\{ \sum_{k=1}^{n-1} \frac{(1 - \tilde{\gamma})(\tilde{\gamma} - \gamma)^{k-1}}{(1 - \gamma)^k} \binom{n-1}{n-k} \gamma^{n-k}(1 - \gamma)^k \right\} + \alpha_n(1 - \gamma)^n \\
&= (1 - \tilde{\gamma}) \left\{ \sum_{k=1}^{n-1} \binom{n-1}{n-k} (\tilde{\gamma} - \gamma)^{k-1}\gamma^{n-k} \right\} + \alpha_n(1 - \gamma)^n \\
&= (1 - \tilde{\gamma}) \left\{ \sum_{k=1}^{n} \binom{n-1}{n-k} (\tilde{\gamma} - \gamma)^{k-1}\gamma^{n-k} \right\} - (1 - \tilde{\gamma})(\tilde{\gamma} - \gamma)^{n-1} + \alpha_n(1 - \gamma)^n \\
&= (1 - \tilde{\gamma})\tilde{\gamma}^{n-1} - (1 - \tilde{\gamma})(\tilde{\gamma} - \gamma)^{n-1} + \alpha_n(1 - \gamma)^n
\end{aligned}
$$

Solving for α_n gives

$$\alpha_n = \frac{(1 - \tilde{\gamma})(\tilde{\gamma} - \gamma)^{n-1}}{(1 - \gamma)^n}$$

as desired. $\square$

This reweighting scheme has two special cases of interest. A standard single-step model, with $\gamma = 0$, yields $\alpha_n = (1 - \tilde{\gamma})\tilde{\gamma}^{n-1}$. These weights are familiar from the definition of the discounted state occupancy in terms of a per-timestep mixture (Equation 2.2). Setting $\gamma = \tilde{\gamma}$ yields $\alpha_n = 0^{n-1}$, or a weight of 1 on the first step and 0 on all subsequent steps.[1] This result is also expected: when the model discount matches the target discount, only a single forward pass of the model is required.

Figure 3.2 visually depicts the reweighting scheme and the number of steps required for truncated model rollouts to approximate the distribution induced by a larger discount. There is a natural tradeoff with γ-models: the higher γ is, the fewer model steps are needed to make long-horizon predictions, reducing model-based compounding prediction errors (Asadi et al., 2019; Janner et al., 2019). However, increasing γ transforms what would normally

---

[1] We define 0^0 as $\lim_{x \to 0} x^x = 1$.

(a) Reweighting $\gamma = 0.95$ predictions to $\widetilde{\gamma} = 0.99$ **(b)**   Required model rollout lengths
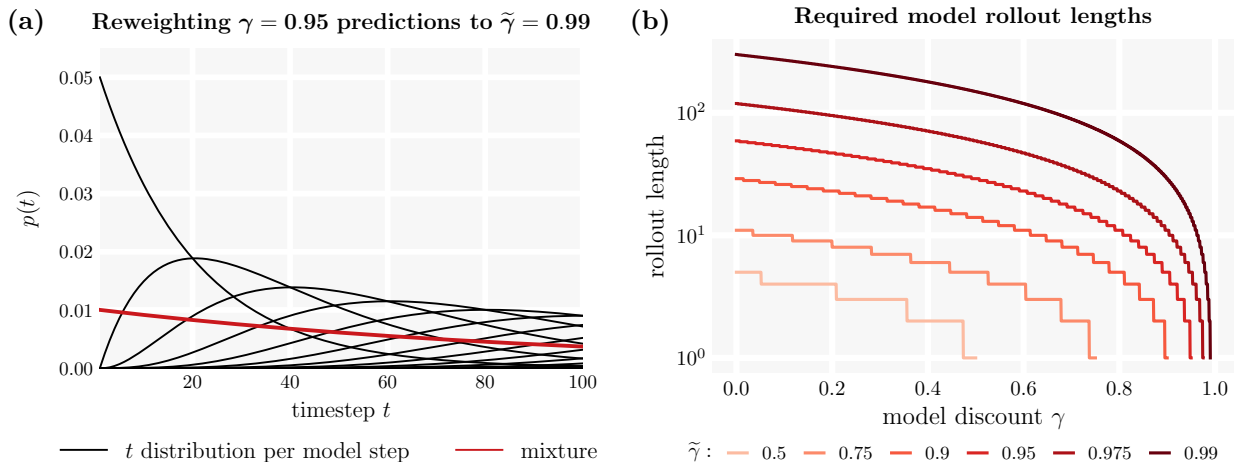
Figure 3.2: **(Rollouts with probabilistic horizons) (a)** The first step from a γ-model samples states at timesteps distributed according to a geometric distribution with parameter $1 - \gamma$; all subsequent steps have a negative binomial timestep distribution stemming from the sum of independent geometric random variables. When these steps are reweighted according to Theorem 1, the resulting distribution follows a geometric distribution with smaller parameter (corresponding to a larger discount value $\tilde{\gamma}$). **(b)** The number of steps needed to recover 95% of the probability mass from distributions induced by various target discounts $\tilde{\gamma}$ for all valid model discounts γ. When using a standard single-step model, corresponding to the case of $\gamma = 0$, a 299-step model rollout is required to reweight to a discount of $\tilde{\gamma} = 0.99$.

be a standard maximum likelihood problem (in the case of single-step models) into one resembling approximate dynamic programming (with a model bootstrap), leading to model-free bootstrap error accumulation (Kumar et al., 2019a). The primary distinction is whether this accumulation occurs during training, when the work of sampling from the occupancy μ is being amortized, or during "testing", when the model is being used for rollouts. While this horizon-based error compounding cannot be eliminated entirely, γ-models allow for a continuous interpolation between the two extremes.

γ-Model-Based Value Expansion

We now turn our attention from prediction with γ-models to value estimation for control. In tabular domains, the state-action value function can be decomposed as the inner product between the successor representation M and the vector of per-state rewards (Gershman, 2018). Taking care to account for the normalization from the equivalence in Proposition 1, we can similarly estimate the Q function as the expectation of reward under states sampled

14

from the γ-model:

$$
\begin{aligned}
Q(\mathbf{s}_t, \mathbf{a}_t; \gamma) &= \sum_{\Delta t=1}^{\infty} \gamma^{\Delta t-1} \int_{\mathcal{S}} r(\mathbf{s}_e) p(\mathbf{s}_{t+\Delta t} = \mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t, \pi) \mathrm{d}\mathbf{s}_e \\
&= \int_{\mathcal{S}} r(\mathbf{s}_e) \sum_{\Delta t=1}^{\infty} \gamma^{\Delta t-1} p(\mathbf{s}_{t+\Delta t} = \mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t, \pi) \mathrm{d}\mathbf{s}_e \\
&= \frac{1}{1-\gamma} \mathbb{E}_{\mathbf{s}_e \sim \mu(\cdot|\mathbf{s}_t, \mathbf{a}_t; \gamma)} \left[r(\mathbf{s}_e) \right]
\end{aligned}
\tag{3.6}
$$

This relation suggests a model-based reinforcement learning algorithm in which Q-values are estimated by a γ-model without the need for sequential model-based rollouts. However, in some cases it may be practically difficult to train a generative γ-model with discount as large as that of a discriminative Q-function. While one option is to chain together γ-model steps as in Section 3.4, an alternative solution often effective with single-step models is to combine short-term value estimates from a truncated model rollout with a terminal model-free value prediction:

$$
V_{\mathrm{MVE}}(\mathbf{s}_t; \tilde{\gamma}) = \sum_{n=1}^{H} \tilde{\gamma}^{n-1} r(\mathbf{s}_{t+n}) + \tilde{\gamma}^{H} V(\mathbf{s}_{t+H}; \tilde{\gamma}).
$$

This hybrid estimator is referred to as a model-based value expansion (MVE; Feinberg et al. 2018). There is a hard transition between the model-based and model-free value estimation in MVE, occuring at the model horizon H. We may replace the single-step model with a γ-model for a similar estimator in which there is a probabilistic prediction horizon, and as a result a gradual transition.

Theorem 2. *For $\tilde{\gamma} > \gamma$, $V(\mathbf{s}_t; \tilde{\gamma})$ may be decomposed as a weighted average of H γ-model steps and a terminal value estimation. We denote this the $\boldsymbol{\gamma}$-MVE estimator:*

$$
\hat{V}_{\gamma\text{-MVE}}(\mathbf{s}_t; \tilde{\gamma}) = \frac{1}{1-\tilde{\gamma}} \sum_{n=1}^{H} \alpha_n \mathbb{E}_{\mathbf{s}_e \sim \mu_n(\cdot|\mathbf{s}_t; \gamma)} \left[r(\mathbf{s}_e) \right] + \left(\frac{\tilde{\gamma} - \gamma}{1 - \gamma} \right)^{H} \mathbb{E}_{\mathbf{s}_e \sim \mu_H(\cdot|\mathbf{s}_t; \gamma)} \left[V(\mathbf{s}_e; \tilde{\gamma}) \right].
$$

Proof.

$$V(\mathbf{s}_t; \tilde{\gamma}) = \frac{1}{1 - \tilde{\gamma}} \mathbb{E}_{\mathbf{s}_e \sim \mu(\cdot | \mathbf{s}_t; \tilde{\gamma})} \left[r(\mathbf{s}_e) \right]$$

$$= \frac{1}{1 - \tilde{\gamma}} \sum_{n=1}^{\infty} \alpha_n \mathbb{E}_{\mathbf{s}_e \sim \mu_n(\cdot | \mathbf{s}_t; \gamma)} \left[r(\mathbf{s}_e) \right]$$

$$= \underbrace{\frac{1}{1 - \tilde{\gamma}} \sum_{n=1}^{H} \alpha_n \mathbb{E}_{\mathbf{s}_e \sim \mu_n(\cdot | \mathbf{s}_t; \gamma)} \left[r(\mathbf{s}_e) \right]}_{\textcircled{1}} + \underbrace{\frac{1}{1 - \tilde{\gamma}} \sum_{n=H+1}^{\infty} \alpha_n \mathbb{E}_{\mathbf{s}_e \sim \mu_n(\cdot | \mathbf{s}_t; \gamma)} \left[r(\mathbf{s}_e) \right]}_{\textcircled{2}}. \quad (3.7)$$

The second equality rewrites an expectation over a $\tilde{\gamma}$-model as an expectation over a rollout of a γ-model using step weights α_n from Theorem 1. We recognize $\textcircled{1}$ as the model-based component of the value estimation in γ-MVE. All that remains is to write $\textcircled{2}$ using a terminal value function.

$$\sum_{n=H+1}^{\infty} \alpha_n \mathbb{E}_{\mathbf{s}_e \sim \mu_n(\cdot | \mathbf{s}_t; \gamma)} \left[r(\mathbf{s}_e) \right] = \sum_{n=1}^{\infty} \alpha_{H+n} \mathbb{E}_{\mathbf{s}_e \sim \mu_{H+n}(\cdot | \mathbf{s}_t; \gamma)} \left[r(\mathbf{s}_e) \right]$$

$$= \left(\frac{\tilde{\gamma} - \gamma}{1 - \gamma} \right)^H \mathbb{E}_{\mathbf{s}_H \sim \mu_H(\cdot | \mathbf{s}_t; \gamma)} \left[\sum_{n=1}^{\infty} \alpha_n \mathbb{E}_{\mathbf{s}_e \sim \mu_n(\cdot | \mathbf{s}_H; \gamma)} \left[r(\mathbf{s}_e) \right] \right]$$

$$= \left(\frac{\tilde{\gamma} - \gamma}{1 - \gamma} \right)^H \mathbb{E}_{\mathbf{s}_H \sim \mu_H(\cdot | \mathbf{s}_t; \gamma)} \left[\mathbb{E}_{\mathbf{s}_e \sim \mu(\cdot | \mathbf{s}_H; \tilde{\gamma})} \left[r(\mathbf{s}_e) \right] \right]$$

$$= (1 - \tilde{\gamma}) \left(\frac{\tilde{\gamma} - \gamma}{1 - \gamma} \right)^H \mathbb{E}_{\mathbf{s}_e \sim \mu_H(\cdot | \mathbf{s}_t; \gamma)} \left[V(\mathbf{s}_e; \tilde{\gamma}) \right] \quad (3.8)$$

The second equality uses $\alpha_{H+n} = \left(\frac{\tilde{\gamma} - \gamma}{1 - \gamma} \right)^H \alpha_n$ and the time-invariance of $G^{(n)}$ with respect to its conditioning state. Plugging Equation 3.8 into Equation 3.7 gives:

$$V(\mathbf{s}_t; \tilde{\gamma}) = \frac{1}{1 - \tilde{\gamma}} \sum_{n=1}^{H} \alpha_n \mathbb{E}_{\mathbf{s}_e \sim \mu_n(\cdot | \mathbf{s}_t; \gamma)} \left[r(\mathbf{s}_e) \right] + \left(\frac{\tilde{\gamma} - \gamma}{1 - \gamma} \right)^H \mathbb{E}_{\mathbf{s}_e \sim \mu_H(\cdot | \mathbf{s}_t; \gamma)} \left[V(\mathbf{s}_e; \tilde{\gamma}) \right].$$

$$\square$$

Remark 1. Using Lemma 1 from Appendix A.1 to substitute $1 - \sum_{n=1}^{H} \alpha_n$ in place of $\left(\frac{\tilde{\gamma} - \gamma}{1 - \gamma} \right)^H$ clarifies the interpretation of $V(\mathbf{s}_t; \tilde{\gamma})$ as a weighted average over H γ-model steps and a terminal value function. Because the mixture weights must sum to 1, it is unsurprising that the weight on the terminal value function turned out to be $\left(\frac{\tilde{\gamma} - \gamma}{1 - \gamma} \right)^H = 1 - \sum_{n=1}^{H} \alpha_n$.

Algorithm 1 γ-**model based value expansion**

1: **Input** γ: model discount, $\tilde{\gamma}$: value discount, λ : step size
2: **Initialize** μ_θ : γ-model generator
3: **Initialize** Q_ω : Q-function, V_ξ : value function, π_ψ : policy, $\mathcal{D}$: replay buffer
4: **for** each iteration **do**
5:     **for** each environment step **do**
6:         $\mathbf{a}_t \sim \pi_\psi(\cdot \mid \mathbf{s}_t)$
7:         $\mathbf{s}_{t+1} \sim p(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)$
8:         $\mathbf{r}_t = r(\mathbf{s}_t, \mathbf{a}_t)$
9:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}\}$
10:     **end for**
11:     **for** each gradient step **do**
12:         Sample transitions $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1})$ from $\mathcal{D}$
13:         Update μ_θ to Algorithm 2 or 3
14:         Compute $V_{\gamma-\mathrm{MVE}}(\mathbf{s}_{t+1})$ according to Theorem 2
15:         Update Q-function parameters:
$$\omega \leftarrow \omega - \lambda \nabla_\omega \tfrac{1}{2} \left(Q_\omega(\mathbf{s}_t, \mathbf{a}_t) - (\mathbf{r}_t + \tilde{\gamma} V_{\gamma-\mathrm{MVE}}(\mathbf{s}_{t+1})) \right)^2$$
16:         Update value function parameters:
$$\xi \leftarrow \xi - \lambda \nabla_\xi \tfrac{1}{2} \left(V_\xi(\mathbf{s}_t) - \mathbb{E}_{\mathbf{a}\sim\pi_\psi(\cdot|\mathbf{s}_t)} \left[Q_\omega(\mathbf{s}_t, \mathbf{a}) - \log \pi_\psi(\mathbf{a} \mid \mathbf{s}_t) \right] \right)^2$$
17:         Update policy parameters:
$$\psi \leftarrow \psi - \lambda \nabla_\psi \mathbb{E}_{\mathbf{a}\sim\pi_\psi(\cdot|\mathbf{s}_t)} \left[\log \pi_\psi(\mathbf{a} \mid \mathbf{s}_t) - Q_\omega(\mathbf{s}_t, \mathbf{a}) \right]$$
18:     **end for**
19: **end for**

Remark 2. Setting $\gamma = 0$ recovers standard MVE with a single-step model, as the weights on the model steps simplify to $\alpha_n = (1 - \tilde{\gamma})(\tilde{\gamma} - \gamma)^{n-1}$ and the weight on the terminal value function simplifies to $\tilde{\gamma}^H$.

The γ-MVE estimator allows us to perform γ-model-based rollouts with horizon H, reweight the samples from this rollout by solving for weights α_n given a desired discount $\tilde{\gamma} > \gamma$, and correct for the truncation error stemming from the finite rollout length using a terminal value function with discount $\tilde{\gamma}$. As expected, MVE is a special case of γ-MVE, as can be verified by considering the weights corresponding to $\gamma = 0$ described in Section 3.4. This estimator, along with the simpler value estimation in Equation 3.6, highlights the fact that it is not necessary to have timesteps associated with states in order to use predictions for decision-making. Pseudocode for an actor-critic algorithm using the γ-MVE estimator is provided Algorithm 1.

3.5 Practical Training of γ-Models

Because γ-model training differs from standard dynamics modeling primarily in the bootstrapped target distribution and not in the model parameterization, γ-models are in principle compatible with any generative modeling framework. We focus on two representative scenarios, differing in whether the generative model class used to instantiate the γ-model allows for tractable density evaluation.

Training without density evaluation. When the γ-model parameterization does not allow for tractable density evaluation, we minimize a bootstrapped f-divergence according to $\mathcal{L}_1$ (Equation 3.2) using only samples from the model. The generative adversarial framework provides a convenient way to train a parametric generator by minimizing an f-divergence of choice given only samples from a target distribution p_{targ} and the ability to sample from the generator (Goodfellow et al., 2014a; Nowozin et al., 2016). In the case of bootstrapped maximum likelihood problems, our target distribution is induced by the model itself (alongside a single-step transition distribution), meaning that we only need sample access to our γ-model in order to train μ_θ as a generative adversarial network (GAN).

Introducing an auxiliary discriminator D_ϕ and selecting the Jensen-Shannon divergence as our f-divergence, we can reformulate minimization of the original objective $\mathcal{L}_1$ as a saddle-point optimization over the following objective:

$$\hat{\mathcal{L}}_1(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\mathbf{s}_e^+ \sim p_{\mathrm{targ}}(\cdot | \mathbf{s}_t, \mathbf{a}_t)} \left[\log D_\phi(\mathbf{s}_e^+ \mid \mathbf{s}_t, \mathbf{a}_t) \right] + \mathbb{E}_{\mathbf{s}_e^- \sim \mu_\theta(\cdot | \mathbf{s}_t, \mathbf{a}_t)} \left[\log(1 - D_\phi(\mathbf{s}_e^- \mid \mathbf{s}_t, \mathbf{a}_t)) \right],$$

which is minimized over μ_θ and maximized over D_ϕ. As in $\mathcal{L}_1$, p_{targ} refers to the bootstrapped target distribution in Equation 3.1. In this formulation, μ_θ produces samples by virtue of a deterministic mapping of a random input vector $\mathbf{z} \sim \mathcal{N}(0, I)$ and conditioning information $(\mathbf{s}_t, \mathbf{a}_t)$. Other choices of f-divergence may be instantiated by different choices of activation function (Nowozin et al., 2016).

Training with density evaluation. When the γ-model permits density evaluation, we may bypass saddle point approximations to an f-divergence and directly regress to target density values, as in objective $\mathcal{L}_2$ (Equation 3.3). This is a natural choice when the γ-model is instantiated as a conditional normalizing flow (Rezende & Mohamed, 2015). Evaluating target values of the form

$$T(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{s}_e) = \log \left((1 - \gamma)p(\mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t) + \gamma \mu_\theta(\mathbf{s}_e \mid \mathbf{s}_{t+1}) \right)$$

requires density evaluation of not only our γ-model, but also the single-step transition distribution. There are two options for estimating the single-step densities: (1) a single-step

---

Algorithm 2 γ-model training without density evaluation

1: **Input** $\mathcal{D}$: dataset of transitions, π : policy, λ : step size, τ : delay parameter
2: Initialize parameter vectors $\theta, \bar{\theta}, \phi$
3: **while** not converged **do**
4:      Sample transitions $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ from $\mathcal{D}$ and actions $\mathbf{a}_{t+1} \sim \pi(\cdot \mid \mathbf{s}_{t+1})$
5:      Sample from bootstrapped target $\mathbf{s}_e^+ \sim (1-\gamma)\delta_{\mathbf{s}_{t+1}} + \gamma\mu_{\bar{\theta}}(\cdot \mid \mathbf{s}_{t+1}, \mathbf{a}_{t+1})$
6:      Sample from current model $\mathbf{s}_e^- \sim \mu_\theta(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)$
7:      Evaluate objective $\mathcal{L} = \log D_\phi(\mathbf{s}_e^+ \mid \mathbf{s}_t, \mathbf{a}_t) + \log(1 - D_\phi(\mathbf{s}_e^- \mid \mathbf{s}_t, \mathbf{a}_t))$
8:      Update model parameters $\theta \leftarrow \theta - \lambda\nabla_\theta\mathcal{L}; \;\; \phi \leftarrow \phi + \lambda\nabla_\phi\mathcal{L}$
9:      Update target parameters $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$
10: **end while**

Algorithm 3 γ-model training with density evaluation

1: **Input** $\mathcal{D}$: dataset of transitions, π : policy, λ : step size, τ : delay parameter, σ^2 : variance
2: Initialize parameter vectors $\theta, \bar{\theta}$; let f denote the Gaussian pdf
3: **while** not converged **do**
4:      Sample transitions $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ from $\mathcal{D}$ and actions $\mathbf{a}_{t+1} \sim \pi(\cdot \mid \mathbf{s}_{t+1})$
5:      Sample from bootstrapped target $\mathbf{s}_e \sim (1-\gamma)\mathcal{N}(\mathbf{s}_{t+1}, \sigma^2) + \gamma\mu_{\bar{\theta}}(\cdot \mid \mathbf{s}_{t+1}, \mathbf{a}_{t+1})$
6:      Construct target values $T = \log\left((1-\gamma)f(\mathbf{s}_e \mid \mathbf{s}_{t+1}, \sigma^2) + \gamma\mu_{\bar{\theta}}(\mathbf{s}_e \mid \mathbf{s}_{t+1}, \mathbf{a}_{t+1})\right)$
7:      Evaluate objective $\mathcal{L} = \|\log\mu_\theta(\mathbf{s}_e \mid \mathbf{s}_t, \mathbf{a}_t) - T\|_2^2$
8:      Update model parameters $\theta \leftarrow \theta - \lambda\nabla_\theta\mathcal{L}$
9:      Update target parameters $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$
10: **end while**

---

model p_θ may be trained alongside the γ-model μ_θ for the purposes of constructing targets $T(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{s}_e)$, or (2) a simple approximate model may be constructed on the fly from $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ transitions. We found $p_\theta = \mathcal{N}(\mathbf{s}_{t+1}, \sigma^2)$, with σ^2 a constant hyperparameter, to be sufficient.

Stability considerations. To alleviate the instability caused by bootstrapping, we appeal to the standard solution employed in model-free reinforcement learning: decoupling the regression targets from the current model by way of a "delayed" target network (Mnih et al., 2015). In particular, we use a delayed γ-model $\mu_{\bar{\theta}}$ in the bootstrapped target distribution p_{targ}, with the parameters $\bar{\theta}$ given by an exponentially-moving average of previous parameters θ.

We summarize the above scenarios in Algorithms 2 and 3. We isolate model training from data collection and focus on a setting in which a static dataset is provided, but this algorithm

may also be used in a data-collection loop for policy improvement. Further implementation details, including all hyperparameter settings and network architectures, are included in Appendix A.2.

3.6   Experimental Evaluation

Our experimental evaluation is designed to study the viability of γ-models as a replacement of conventional single-step models for long-horizon state prediction and model-based control.
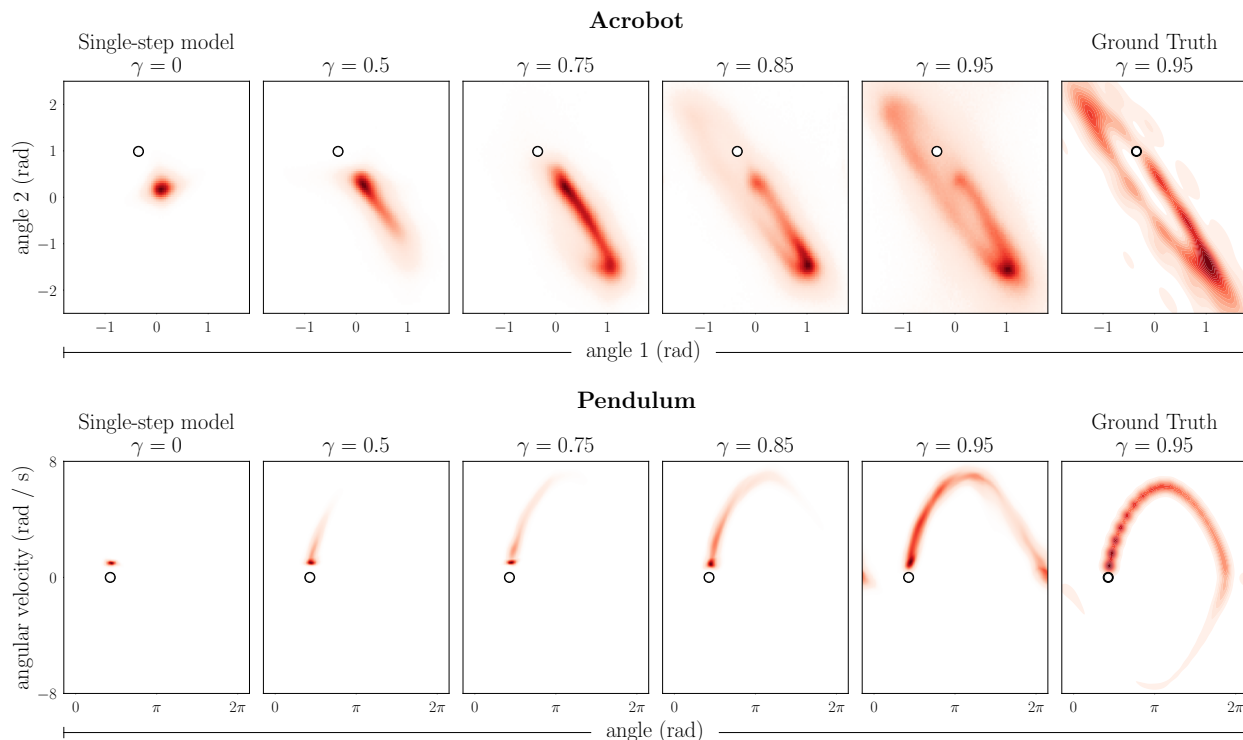
Figure 3.3: (γ-model predictions) Visualization of the predicted distribution from a **single** feedforward pass of normalizing flow γ-models trained with varying discounts γ. The conditioning state $\mathbf{s}_t$ is denoted by ○. The leftmost plots, with $\gamma = 0$, correspond to a single-step model. For comparison, the rightmost plots show a Monte Carlo estimation of the discounted occupancy from 100 environment trajectories.

Prediction

We investigate γ-model predictions as a function of discount in continuous-action versions of two benchmark environments suitable for visualization: acrobot (Sutton, 1996) and pendulum. The training data come from a mixture distribution over all intermediate policies of 200 epochs of optimization with soft-actor critic (SAC; Haarnoja et al. 2018). The final converged policy is used for γ-model training. We refer to Appendix A.2 for implementation and experiment details.

Figure 3.3 shows the predictions of a γ-model trained as a normalizing flow according to Algorithm 3 for five different discounts, ranging from $\gamma = 0$ (a single-step model) to $\gamma = 0.95$. The rightmost column shows the ground truth discounted occupancy corresponding to $\gamma = 0.95$, estimated with Monte Carlo rollouts of the policy. Increasing the discount γ during training has the expected effect of qualitatively increasing the predictive lookahead of a single feedforward pass of the γ-model. We found flow-based γ-models to be more reliable than GAN parameterizations, especially at higher discounts. Corresponding GAN γ-model visualizations can be found in Appendix A.4 for comparison.

Equation 3.6 expresses values as an expectation over a single feedforward pass of a γ-model.
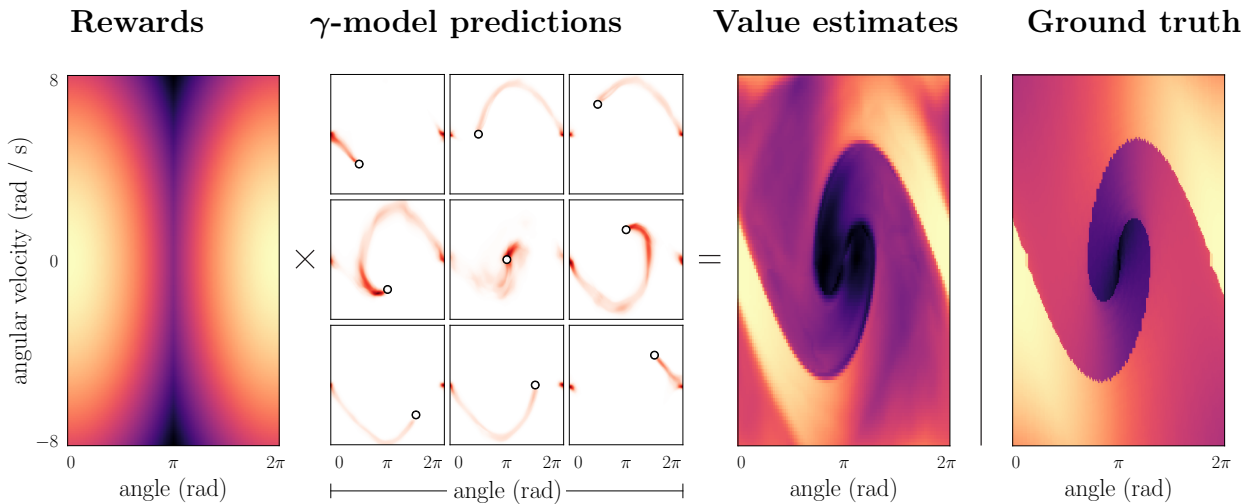
Figure 3.4: (**γ-model value estimation**) Values are expectations of reward over a single feedforward pass of a γ-model (Equation 3.6). We visualize γ-model predictions ($\gamma = 0.99$) from nine starting states, denoted by ○, in the pendulum benchmark environment. Taking the expectation of reward over each of these predicted distributions yields a value estimate for the corresponding conditioning state. The rightmost plot depicts the value map produced by value iteration on a discretization of the same environment for reference.
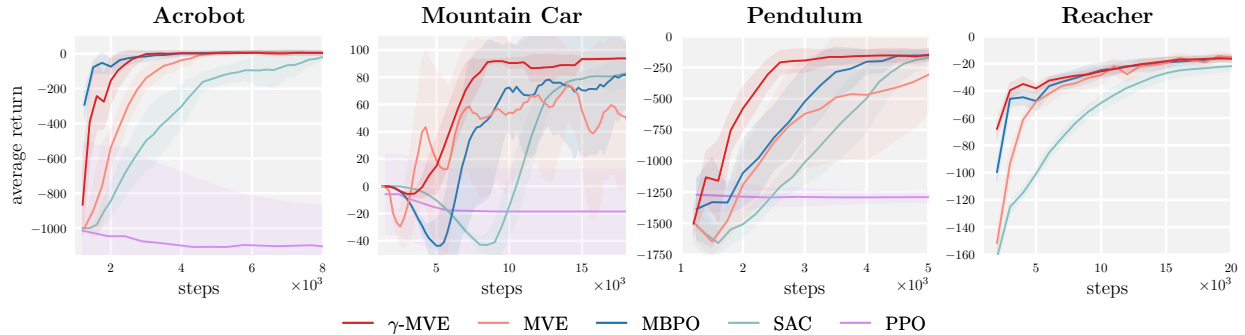
Figure 3.5: **(γ-MVE control performance)** Comparative performance of γ-MVE and four prior reinforcement learning algorithms on continuous control benchmark tasks. γ-MVE retains the asymptotic performance of SAC with sample-efficiency matching that of MBPO. Shaded regions depict standard deviation among 5 seeds.

We visualize this relation in Figure 3.4, which depicts γ-model predictions on the pendulum environment for a discount of $\gamma = 0.99$ and the resulting value map estimated by taking expectations over these predicted state distributions. In comparison, value estimation for the same discount using a single-step model would require 299-step rollouts in order to recover 95% of the probability mass (see Figure 3.2).

Control

To study the utility of the γ-model for model-based reinforcement learning, we use the γ-MVE estimator from Section 3.4 as a drop-in replacement for value estimation in SAC. We compare this approach to the state-of-the-art in model-based and model-free methods, with representative algorithms consisting of SAC, PPO (Schulman et al., 2017), MBPO (Janner et al., 2019), and MVE (Feinberg et al., 2018). In γ-MVE, we use a model discount of $\gamma = 0.8$, a value discount of $\tilde{\gamma} = 0.99$ and a single model step ($n = 1$). We use a model rollout length of 5 in MVE such that it has an effective horizon identical to that of γ-MVE. Other hyperparameter settings can once again be found in Appendix A.2; details regarding the evaluation environments can be found in Appendix A.3. Figure 3.5 shows learning curves for all methods. We find that γ-MVE converges faster than prior algorithms, twice as quickly as SAC, while retaining their asymptotic performance.

3.7 Discussion

We have introduced a new class of predictive model, a γ-model, that is a hybrid between standard model-free and model-based mechanisms. It is policy-conditioned and infinite-horizon, like a value function, but independent of reward, like a standard single-step model. This new formulation of infinite-horizon prediction allows us to generalize the procedures integral to model-based control, yielding new variants of model rollouts and model-based value estimation. Our experimental evaluation shows that, on tasks with low to moderate dimensionality, our method learns accurate long-horizon predictive distributions without sequential rollouts and can be incorporated into standard model-based reinforcement learning methods to produce results that are competitive with state-of-the-art algorithms.

However, scaling up this framework to more complex tasks, including high-dimensional continuous control problems and tasks with image observations, presents a number of additional challenges. These challenges are largely those of generative modeling; whereas temporal differences algorithms are conventionally used to estimate expectations of scalar random variables, we are here employing them to estimate high-dimensional joint distributions. Unsurprisingly, this approach can fall short above a threshold dimensionality (in our experience, above 10) or discount factor (above 0.99), as these both increase the complexity of the distribution in question. In the next chapter, we investigate whether the language modeling toolbox can be used to address this limitation directly and provide effective generative modeling solutions for the reinforcement learning problem setting.

4

Reinforcement Learning as Sequence Modeling

4.1 Introduction

The standard treatment of reinforcement learning relies on decomposing a long-horizon problem into smaller, more local subproblems. In model-free algorithms, this takes the form of the principle of optimality (Bellman, 1957), a recursion that leads naturally to the class of dynamic programming methods like Q-learning. In model-based algorithms, this decomposition takes the form of single-step predictive models, which reduce the problem of predicting high-dimensional, policy-dependent state trajectories to that of estimating a comparatively simpler, policy-agnostic transition distribution. As seen in Chapter 3, these two approaches constitute the endpoints of a spectrum, and it is possible to design methods that interpolate between them.

However, we can also view reinforcement learning as analogous to a sequence generation problem, with the goal being to produce a sequence of actions that, when enacted in an environment, will yield a sequence of high rewards. In this chapter, we consider the logical extreme of this analogy: does the toolbox of contemporary sequence modeling itself provide a viable reinforcement learning algorithm? We investigate this question by treating trajectories as unstructured sequences of states, actions, and rewards. We model the distribution of these trajectories using a Transformer architecture (Vaswani et al., 2017), the current tool of choice for capturing long-horizon dependencies. In place of the trajectory optimizers common in model-based control, we use beam search (Reddy, 1977), a heuristic decoding scheme ubiquitous in natural language processing, as a planning algorithm.

Posing reinforcement learning, and more broadly data-driven control, as a sequence modeling problem handles many of the considerations that typically require distinct solutions: actor-critic algorithms require separate actors and critics, model-based algorithms require predictive dynamics models, and offline reinforcement learning methods often require estimation of the
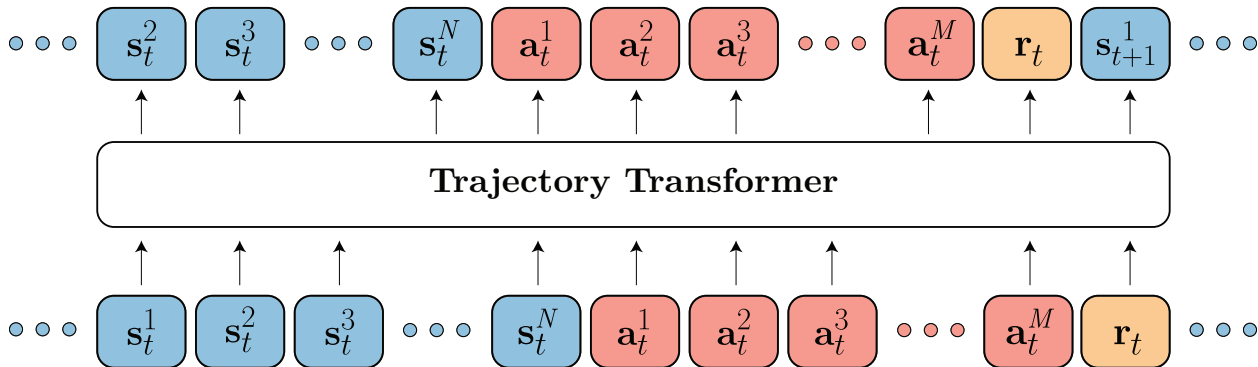
Figure 4.1: **(Trajectory Transformer architecture)** The Trajectory Transformer trains on sequences of (autoregressively discretized) states, actions, and rewards. Planning with the Trajectory Transformer mirrors the sampling procedure used to generate sequences from a language model.

behavior policy (Fujimoto et al., 2019). These components estimate different densities or distributions, such as that over actions in the case of actors and behavior policies, or that over states in the case of dynamics models. Even value functions can be viewed as performing inference in a graphical model with auxiliary optimality variables, amounting to estimation of the distribution over future rewards (Levine, 2018). All of these problems can be unified under a single sequence model, which treats states, actions, and rewards as simply a stream of data. The advantage of this perspective is that high-capacity sequence model architectures can be brought to bear on the problem, resulting in a more streamlined approach that could benefit from the same scalability underlying large-scale unsupervised learning results (Brown et al., 2020).

We refer to our model as the Trajectory Transformer (Figure 5.2) and evaluate it in the offline regime so as to be able to make use of large amounts of prior interaction data. The Trajectory Transformer is a substantially more reliable long-horizon predictor than conventional dynamics models, even in Markovian environments for which the standard model parameterization is in principle sufficient. When decoded with a modified beam search procedure that biases trajectory samples according to their cumulative reward, the Trajectory Transformer attains results on offline reinforcement learning benchmarks that are competitive with the best prior methods designed specifically for that setting. Additionally, we describe how variations of the same decoding procedure yield a model-based imitation learning method, a goal-reaching method, and, when combined with dynamic programming, a state-of-the-art planner for sparse-reward, long-horizon tasks. Our results suggest that the algorithms and architectural motifs that have been widely applicable in unsupervised learning carry similar benefits in reinforcement learning.

4.2   Related Work

Recent advances in sequence modeling with deep networks have led to rapid improvement in the effectiveness of such models, from LSTMs and sequence-to-sequence models (Hochreiter & Schmidhuber, 1997; Sutskever et al., 2014) to Transformer architectures with self-attention (Vaswani et al., 2017). In light of this, it is tempting to consider how such sequence models can lead to improved performance in reinforcement learning, which is also concerned with sequential processes (Sutton, 1988). Indeed, a number of prior works have studied applying sequence models of various types to represent components in standard reinforcement learning algorithms, such as policies, value functions, and models (Bakker, 2002; Heess et al., 2015a; Chiappa et al., 2017; Parisotto et al., 2020; Parisotto & Salakhutdinov, 2021; Kumar et al., 2020b). While such works demonstrate the importance of such models for representing memory (Oh et al., 2016), they still rely on standard reinforcement learning algorithmic advances to improve performance. The goal in our work is different: we aim to replace as much of the reinforcement learning pipeline as possible with sequence modeling, so as to produce a simpler method whose effectiveness is determined by the representational capacity of the sequence model rather than algorithmic sophistication.

Estimation of probability distributions and densities arises in many places in learning-based control. This is most obvious in model-based reinforcement learning, where it is used to train predictive models that can then be used for planning or policy learning (Sutton, 1990; Silver et al., 2008; Fairbank, 2008; Deisenroth & Rasmussen, 2011; Lampe & Riedmiller, 2014; Heess et al., 2015b; Janner et al., 2020; Amos et al., 2021). However, it also figures heavily in offline reinforcement learning, where it is used to estimate conditional distributions over actions that serve to constrain the learned policy to avoid out-of-distribution behavior that is not supported under the dataset (Fujimoto et al., 2019; Kumar et al., 2019a; Ghasemipour et al., 2020); imitation learning, where it is used to fit an expert's actions to obtain a policy (Ross & Bagnell, 2010; Ross et al., 2011); and other areas such as hierarchical reinforcement learning (Peng et al., 2017; Co-Reyes et al., 2018; Jiang et al., 2019). In our method, we train a single high-capacity sequence model to represent the joint distribution over sequences of states, actions, and rewards. This serves as *both* a predictive model *and* a behavior policy (for imitation) or behavior constraint (for offline reinforcement learning).

Our approach to reinforcement learning is most closely related to prior model-based methods that plan with a learned model (Chua et al., 2018; Wang & Ba, 2020). However, while these prior methods typically require additional machinery to work well, such as ensembles in the online setting (Kurutach et al., 2018; Buckman et al., 2018a; Malik et al., 2019) or conservatism mechanisms in the offline setting (Yu et al., 2020; Kidambi et al., 2020; Argenson & Dulac-Arnold, 2021), our method does not require explicit handling of these components. Modeling the states and actions jointly already provides a bias toward generating in-distribution actions, which avoids the need for explicit pessimism (Fujimoto et al., 2019;

Kumar et al., 2019a; Ghasemipour et al., 2020; Nair et al., 2020; Jin et al., 2021; Yin et al., 2021; Dadashi et al., 2021). Our method also differs from most prior model-based algorithms in the dynamics model architecture used, with fully-connected networks parameterizing diagonal-covariance Gaussian distributions being a common choice (Chua et al., 2018), though recent work has highlighted the effectiveness of autoregressive state prediction (Zhang et al., 2021) like that used by the Trajectory Transformer. In the context of recently proposed offline reinforcement learning algorithms, our method can be interpreted as a combination of model-based reinforcement learning and policy constraints (Kumar et al., 2019a; Wu et al., 2019), though our approach does not require introducing such constraints explicitly. In the context of model-free reinforcement learning, our method also resembles recently proposed work on goal relabeling (Andrychowicz et al., 2017; Rauber et al., 2019; Ghosh et al., 2021; Paster et al., 2021) and reward conditioning (Schmidhuber, 2019; Srivastava et al., 2019; Kumar et al., 2019b) to reinterpret all past experience as useful demonstrations with proper contextualization.

Concurrently with our work, Chen et al. (2021b) also proposed a reinforcement learning approach centered around sequence prediction, focusing on reward conditioning as opposed to the beam-search-based planning used by the Trajectory Transformer. Their work further supports the possibility that a high-capacity sequence model can be applied to reinforcement learning problems without the need for the components usually associated with reinforcement learning algorithms.

4.3 Reinforcement Learning as Sequence Modeling

In this section, we describe the training procedure for our sequence model and discuss how it can be used for control. We refer to the model as the Trajectory Transformer for brevity, but emphasize that at the implementation level, both our model and search strategy are nearly identical to those common in natural language processing. As a result, modeling considerations are concerned less with architecture design and more with how to represent trajectory data – potentially consisting of continuous states and actions – for processing by a discrete-token architecture (Radford et al., 2018).

Trajectory Transformer

At the core of our approach is the treatment of trajectory data as an unstructured sequence for modeling by a Transformer architecture. A trajectory τ consists of T states, actions, and scalar rewards:

$$\tau = \big(\mathbf{s}_1, \mathbf{a}_1, r_1, \mathbf{s}_2, \mathbf{a}_2, r_2, \ldots, \mathbf{s}_T, \mathbf{a}_T, r_T\big).$$

In the event of continuous states and actions, we discretize each dimension independently. Assuming N-dimensional states and M-dimensional actions, this turns τ into sequence of length $T(N + M + 1)$:

$$\boldsymbol{\tau} = \left(\dots, s_t^1, s_t^2, \dots, s_t^N, a_t^1, a_t^2, \dots, a_t^M, r_t, \dots \right) \qquad t = 1, \dots, T.$$

Subscripts on all tokens denote timestep and superscripts on states and actions denote dimension (*i.e.*, s_t^i is the ith dimension of the state at time t). While this choice may seem inefficient, it allows us to model the distribution over trajectories with more expressivity without simplifying assumptions such as Gaussian transitions.

We investigate two simple discretization approaches:

1. **Uniform:** All tokens for a given dimension correspond to a fixed width of the original continuous space. Assuming a per-dimension vocabulary size of V, the tokens for state dimension i cover uniformly-spaced intervals of width $(\max \mathbf{s}^i - \min \mathbf{s}^i)/V$.

2. **Quantile:** All tokens for a given dimension account for an equal amount of probability mass under the empirical data distribution; each token accounts for 1 out of every V data points in the training set.

Uniform discretization has the advantage that it retains information about Euclidean distance in the original continuous space, which may be more reflective of the structure of a problem than the training data distribution. However, outliers in the data may have outsize effects on the discretization size, leaving many tokens corresponding to zero training points. The quantile discretization scheme ensures that all tokens are represented in the data. We compare the two empirically in Section 4.4.

Our model is a Transformer decoder mirroring the GPT architecture (Radford et al., 2018). We use a smaller architecture than those typically used in large-scale language modeling, consisting of four layers and four self-attention heads. (A full architectural description is provided in Appendix B.1.) Training is performed with the standard teacher-forcing procedure (Williams & Zipser, 1989) used to train sequence models. Denoting the parameters of the Trajectory Transformer as θ and induced conditional probabilities as P_θ, the objective maximized during training is:

$$\mathcal{L}(\tau) = \sum_{t=1}^{T} \left(\sum_{i=1}^{N} \log P_\theta \left(s_t^i \mid \mathbf{s}_t^{<i}, \boldsymbol{\tau}_{<t} \right) + \sum_{j=1}^{M} \log P_\theta \left(a_t^j \mid \mathbf{a}_t^{<j}, \mathbf{s}_t, \boldsymbol{\tau}_{<t} \right) + \log P_\theta \left(r_t \mid \mathbf{a}_t, \mathbf{s}_t, \boldsymbol{\tau}_{<t} \right) \right),$$

in which we use $\boldsymbol{\tau}_{<t}$ to denote a trajectory from timesteps 0 through $t-1$, $\mathbf{s}_t^{<i}$ to denote dimensions 0 through $i-1$ of the state at timestep t, and similarly for $\mathbf{a}_t^{<j}$. We use the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 2.5×10^{-4} to train parameters θ.

Algorithm 4 Beam search

1: **Require** Input sequence $\mathbf{x}$, vocabulary $\mathcal{V}$, sequence length T, beam width B
2: **Initialize** $Y_0 = \{\ (\ )\ \}$
3: **for** $t = 1, \ldots, T$ **do**
4:     $\mathcal{C}_t \leftarrow \{\mathbf{y}_{t-1} \circ y \mid \mathbf{y}_{t-1} \in Y_{t-1} \text{ and } y \in \mathcal{V}\}$ `// candidate single-token extensions`
5:     $Y_t \leftarrow \arg\max Y \subseteq \mathcal{C}_t, |Y| = B \log P_\theta(Y \mid \mathbf{x})$ `// B most likely sequences from`
       `candidates`
6: **end for**
7: **Return** $\arg\max \mathbf{y} \in Y_T \log P_\theta(\mathbf{y} \mid \mathbf{x})$

Planning with Beam Search

We now describe how sequence generation with the Trajectory Transformer can be repurposed for control, focusing on three settings: imitation learning, goal-conditioned reinforcement learning, and offline reinforcement learning. These settings are listed in increasing amount of required modification on top of the sequence model decoding approach routinely used in natural language processing.

The core algorithm providing the foundation of our planning techniques, beam search, is described in Algorithm 4 for generic sequences. Following the presentation in (Meister et al., 2020), we have overloaded $\log P_\theta(\cdot \mid \mathbf{x})$ to define the likelihood of a set of sequences in addition to that of a single sequence: $\log P_\theta(Y \mid x) = \sum_{\mathbf{y} \in Y} \log P_\theta(\mathbf{y} \mid \mathbf{x})$. We use ( ) to denote the empty sequence and $\circ$ to represent concatenation.

Imitation learning. When the goal is to reproduce the distribution of trajectories in the training data, we can optimize directly for the probability of a trajectory $\boldsymbol{\tau}$. This situation matches the goal of sequence modeling exactly and as such we may use Algorithm 4 without modification by setting the conditioning input $\mathbf{x}$ to the current state $\mathbf{s}_t$ (and optionally previous history $\boldsymbol{\tau}_{<t}$).

The result of this procedure is a tokenized trajectory $\boldsymbol{\tau}$, beginning from a current state $\mathbf{s}_t$, that has high probability under the data distribution. If the first action $\mathbf{a}_t$ in the sequence is enacted and beam search is repeated, we have a receding horizon-controller. This approach resembles a long-horizon model-based variant of behavior cloning, in which entire trajectories are optimized to match those of a reference behavior instead of only immediate state-conditioned actions. If we set the predicted sequence length to be the action dimension, our approach corresponds exactly to the simplest form of behavior cloning with an autoregressive policy.

Goal-conditioned reinforcement learning. Transformer architectures feature a "causal" attention mask to ensure that predictions only depend on previous tokens in a sequence.

In the context of natural language, this design corresponds to generating sentences in the linear order in which they are spoken as opposed to an ordering reflecting their hierarchical syntactic structure (see, however, Gu et al. 2019 for a discussion of non-left-to-right sentence generation with autoregressive models). In the context of trajectory prediction, this choice instead reflects physical causality, disallowing future events to affect the past. However, the conditional probabilities of the past given the future are still well-defined, allowing us to condition samples not only on the preceding states, actions, and rewards that have already been observed, but also any future context that we wish to occur. If the future context is a state at the end of a trajectory, we decode trajectories with probabilities of the form:

$$P_\theta(s_t^i \mid \mathbf{s}_t^{<i}, \boldsymbol{\tau}_{<t}, \mathbf{s}_T)$$

We can use this directly as a goal-reaching method by conditioning on a desired final state $\mathbf{s}_T$. If we always condition sequences on a final goal state, we may leave the lower-diagonal attention mask intact and simply permute the input trajectory to $\{\mathbf{s}_T, \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_{T-1}\}$. By prepending the goal state to the beginning of a sequence, we ensure that all other predictions may attend to it without modifying the standard attention implementation. This procedure for conditioning resembles prior methods that use supervised learning to train goal-conditioned policies (Ghosh et al., 2021) and is also related to relabeling techniques in model-free reinforcement learning (Andrychowicz et al., 2017). In our framework, it is identical to the standard subroutine in sequence modeling: inferring the most likely sequence given available evidence.

Offline reinforcement learning. The beam search method described in Algorithm 4 optimizes sequences for their probability under the data distribution. By replacing the log-probabilities of transitions with the predicted reward signal, we can use the same Trajectory Transformer and search strategy for reward-maximizing behavior. Appealing to the control as inference graphical model (Levine, 2018), we are in effect replacing a transition's log-probability in beam search with its log-probability of optimality.

Using beam-search as a reward-maximizing procedure has the risk of leading to myopic behavior. To address this issue, we augment each transition in the training trajectories with reward-to-go: $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ and include it as an additional quantity, discretized identically to the others, to be predicted after immediate rewards r_t. During planning, we then have access to value estimates from our model to add to cumulative rewards. While acting greedily with respect to such Monte Carlo value estimates is known to suffer from poor sample complexity and convergence to suboptimal behavior when online data collection is not allowed, we only use this reward-to-go estimate as a heuristic to guide beam search, and hence our method does not require the estimated values to be as accurate as in methods that rely solely on the value estimates to select actions.

In offline RL, reward-to-go estimates are functions of the *behavior* policy that collected the training data and do not, in general, correspond to the values achieved by the Trajectory

Transformer-derived policy. Of course, it is much simpler to learn the value function of the behavior policy than that of the optimal policy, since we can simply use Monte Carlo estimates without relying on Bellman updates. A value function for an improved policy would provide a better search heuristic, though requires invoking the tools of dynamic programming. In Section 4.4 we show that the simple reward-to-go estimates are sufficient for planning with the Trajectory Transformer in many environments, but that improved value functions are useful in the most challenging settings, such as sparse-reward tasks.

Because the Trajectory Transformer predicts reward and reward-to-go only every $N + M + 1$ tokens, we sample all intermediate tokens according to model log-probabilities, as in the imitation learning and goal-reaching settings. More specifically, we sample full transitions $(\mathbf{s}_t, \mathbf{a}_t, r_t, R_t)$ using likelihood-maximizing beam search, treat these transitions as our vocabulary, and filter sequences of transitions by those with the highest cumulative reward plus reward-to-go estimate.

We have taken a sequence-modeling route to what could be described as a fairly simple-looking model-based planning algorithm, in that we sample candidate action sequences, evaluate their effects using a predictive model, and select the reward-maximizing trajectory. This conclusion is in part due to the close relation between sequence modeling and trajectory optimization. There is one dissimilarity, however, that is worth highlighting: by modeling actions jointly with states and sampling them using the same procedure, we can prevent the model from being queried on out-of-distribution actions. The alternative, of treating action sequences as unconstrained optimization variables that do not depend on state (Nagabandi et al., 2018), can more readily lead to model exploitation, as the problem of maximizing reward under a learned model closely resembles that of finding adversarial examples for a classifier (Goodfellow et al., 2014b).

4.4   Experimental Evaluation

Our experimental evaluation focuses on (1) the accuracy of the Trajectory Transformer as a long-horizon predictor compared to standard dynamics model parameterizations and (2) the utility of sequence modeling tools – namely beam search – as a control algorithm in the context of offline reinforcement learning, imitation learning, and goal-reaching.

Model Analysis

We begin by evaluating the Trajectory Transformer as a long-horizon policy-conditioned predictive model. The usual strategy for predicting trajectories given a policy is to rollout with a single-step model, with actions supplied by the policy. Our protocol differs from the
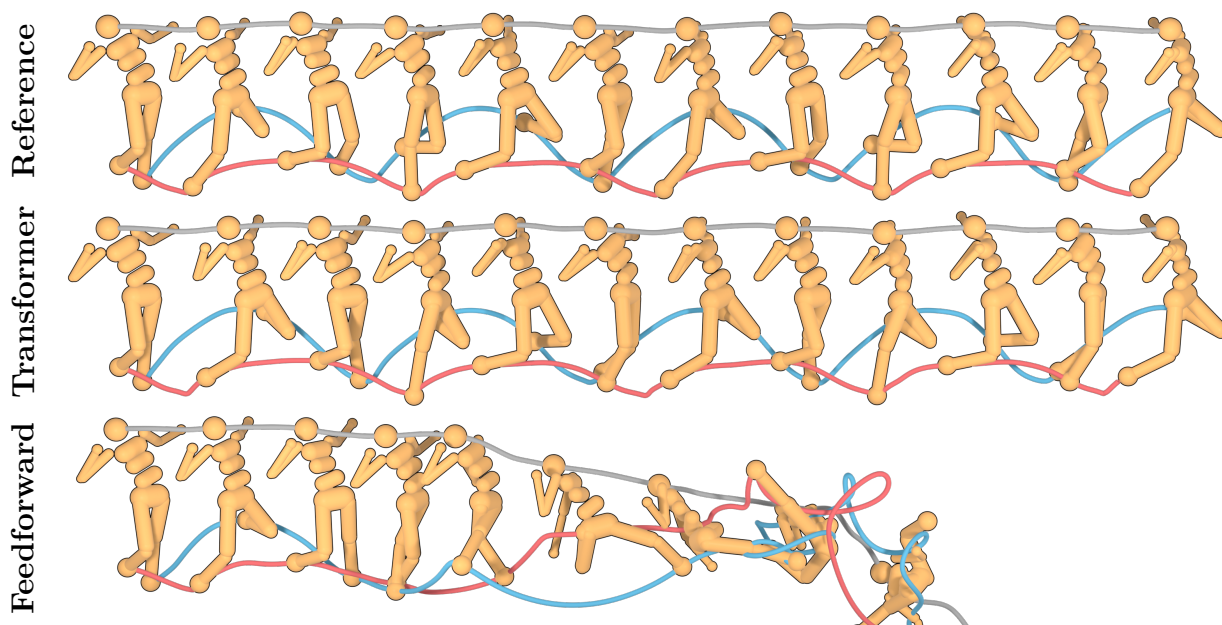
Figure 4.2: **(Transformer prediction visualization)** A qualitative comparison of length-100 trajectories generated by the Trajectory Transformer and a feedforward Gaussian dynamics model from PETS, a state-of-the-art planning algorithm (Chua et al., 2018). Both models were trained on trajectories collected by a single policy, for which a true trajectory is shown for reference. Compounding errors in the single-step model lead to physically implausible predictions, whereas the Transformer-generated trajectory is visually indistinguishable from those produced by the policy acting in the actual environment. The paths of the feet and head are traced through space for depiction of the movement between rendered frames.

standard approach not only in that the model is not Markovian, but also in that it does not require access to a policy to make predictions – the outputs of the policy are modeled alongside the states encountered by that policy. Here, we focus only on the quality of the model's predictions; we use actions predicted by the model for an imitation learning method in the next subsection.

Trajectory predictions. Figure 4.2 depicts a visualization of predicted 100-timestep trajectories from our model after having trained on a dataset collected by a trained humanoid policy. Though model-based methods have been applied to the humanoid task, prior works tend to keep the horizon intentionally short to prevent the accumulation of model errors (Janner et al., 2019; Amos et al., 2021). The reference model is the probabilistic ensemble implementation of PETS (Chua et al., 2018); we tuned the number of models within the ensemble, the number of layers, and layer sizes, but were unable to produce a model that
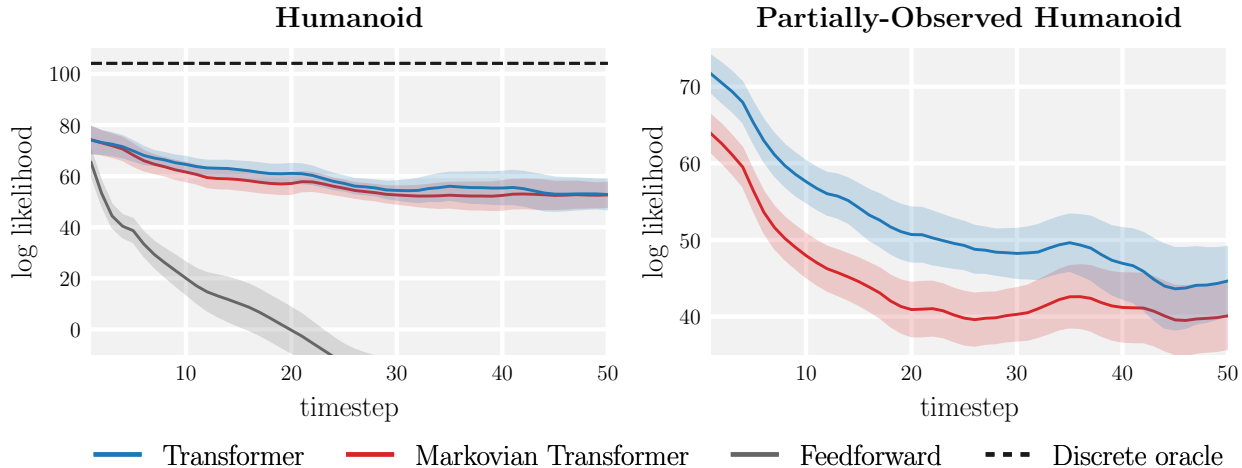
Figure 4.3: **(Compounding model errors)** We compare the accuracy of the Trajectory Transformer (with uniform discretization) to that of the probabilistic feedforward model ensemble (Chua et al., 2018) over the course of a planning horizon in the humanoid environment, corresponding to the trajectories visualized in Figure 4.2. The Trajectory Transformer has substantially better error compounding with respect to prediction horizon than the feedforward model. The discrete oracle is the maximum log likelihood attainable given the discretization size; see Appendix B.2 for a discussion.

predicted accurate sequences for more than a few dozen steps. In contrast, we see that the Trajectory Transformer's long-horizon predictions are substantially more accurate, remaining visually indistinguishable from the ground-truth trajectories even after 100 predicted steps. To our knowledge, no prior model-based reinforcement learning algorithm has demonstrated predicted rollouts of such accuracy and length on tasks of comparable dimensionality.

Error accumulation.   A quantitative account of the same finding is provided in Figure 4.3, in which we evaluate the model's accumulated error versus prediction horizon. Standard predictive models tend to have excellent single-step errors but poor long-horizon accuracy, so instead of evaluating a test-set single-step likelihood, we sample 1000 trajectories from a fixed starting point to estimate the per-timestep state marginal predicted by each model. We then report the likelihood of the states visited by the reference policy on a held-out set of trajectories under these predicted marginals. To evaluate the likelihood under our discretized model, we treat each bin as a uniform distribution over its specified range; by construction, the model assigns zero probability outside of this range.

To better isolate the source of the Transformer's improved accuracy over standard single-step models, we also evaluate a Markovian variant of our same architecture. This ablation has
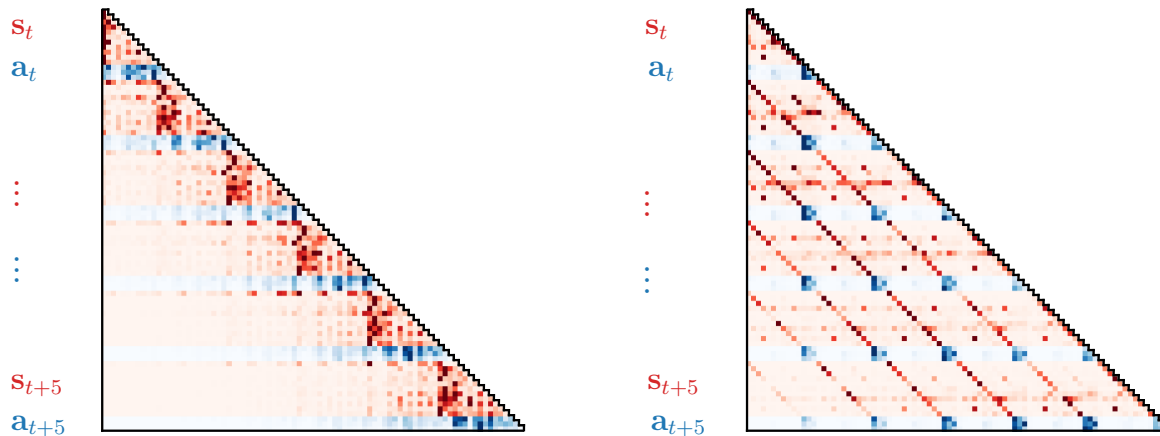
Figure 4.4: **(Attention patterns)** We observe two distinct types of attention masks during trajectory prediction. In the first, both states and actions are dependent primarily on the immediately preceding transition, corresponding to a model that has learned the Markov property. The second strategy has a striated appearance, with state dimensions depending most strongly on the same dimension of multiple previous timesteps. Surprisingly, actions depend more on past actions than they do on past states, reminiscent of the action smoothing used in some trajectory optimization algorithms (Nagabandi et al., 2019). The above masks are produced by a first- and third-layer attention head during sequence prediction on the hopper benchmark; reward dimensions are omitted for this visualization.[1]

a truncated context window that prevents it from attending to more than one timestep in the past. This model performs similarly to the trajectory Transformer on fully-observed environments, suggesting that architecture differences and increased expressivity from the autoregressive state discretization play a large role in the trajectory Transformer's long-horizon accuracy. We construct a partially-observed version of the same humanoid environment, in which each dimension of every state is masked out with 50% probability (Figure 4.3 right), and find that, as expected, the long-horizon conditioning plays a larger role in the model's accuracy in this setting.

Attention patterns. We visualize the attention maps during model predictions in Figure 4.4. We find two primary attention patterns. The first is a discovered Markovian strategy, in which a state prediction attends overwhelmingly to the previous transition. The second is qualitatively striated, with the model attending to specific dimensions in multiple prior states for each state prediction. Simultaneously, the action predictions attend to prior actions more than they do prior states. The action dependencies contrast with the usual formulation of behavior cloning, in which actions are a function of only past states, but is

reminiscent of the action filtering technique used in some planning algorithm to produce smoother action sequences (Nagabandi et al., 2019).

Reinforcement Learning and Control

Offline reinforcement learning. We evaluate the Trajectory Transformer on a number of environments from the D4RL offline benchmark suite (Fu et al., 2020), including the locomotion and AntMaze domains. This evaluation is the most difficult of our control settings, as reward-maximizing behavior is the most qualitatively dissimilar from the types of behavior that are normally associated with unsupervised modeling – namely, imitative behavior. Results for the locomotion environments are shown in Table 4.1. We compare against five other methods spanning other approaches to data-driven control: (1) behavior-regularized actor-critic (BRAC; Wu et al. 2019) and conservative Q-learning (CQL; Kumar et al. 2020a) represent the current state-of-the-art in model-free offline RL; model-based offline planning (MBOP; Argenson & Dulac-Arnold 2021) is the best-performing prior offline trajectory optimization technique; decision transformer (DT; (Chen et al., 2021b)) is a concurrently-developed sequence-modeling approach that uses return-conditioning instead of planning; and behavior-cloning (BC) provides the performance of a pure imitative method.

The Trajectory Transformer performs on par with or better than all prior methods (Table 4.1). The two discretization variants of the Trajectory Transformer, uniform and quantile, perform

Dataset	Environment	BC	MBOP	BRAC	CQL	DT	**TT** (uniform)	**TT** (quantile)
Med-Expert	HalfCheetah	59.9	105.9	41.9	91.6	86.8	40.8 ±2.3	95.0 ±0.2
Med-Expert	Hopper	79.6	55.1	0.9	105.4	107.6	106.0 ±0.28	110.0 ±2.7
Med-Expert	Walker2d	36.6	70.2	81.6	108.8	108.1	91.0 ±2.8	101.9 ±6.8
Medium	HalfCheetah	43.1	44.6	46.3	44.0	42.6	44.0 ±0.31	46.9 ±0.4
Medium	Hopper	63.9	48.8	31.3	58.5	67.6	67.4 ±2.9	61.1 ±3.6
Medium	Walker2d	77.3	41.0	81.1	72.5	74.0	81.3 ±2.1	79.0 ±2.8
Med-Replay	HalfCheetah	4.3	42.3	47.7	45.5	36.6	44.1 ±0.9	41.9 ±2.5
Med-Replay	Hopper	27.6	12.4	0.6	95.0	82.7	99.4 ±3.2	91.5 ±3.6
Med-Replay	Walker2d	36.9	9.7	0.9	77.2	66.6	79.4 ±3.3	82.6 ±6.9
Average		47.7	47.8	36.9	77.6	74.7	72.6	78.9

Table 4.1: **(Offline reinforcement learning)** The Trajectory Transformer (TT) performs on par with or better than the best prior offline reinforcement learning algorithms on D4RL locomotion (v2) tasks. Results for TT variants correspond to the mean and standard error over 15 random seeds (5 independently trained Transformers and 3 trajectories per Transformer). We detail the sources of the performance for other methods in Appendix C.1.
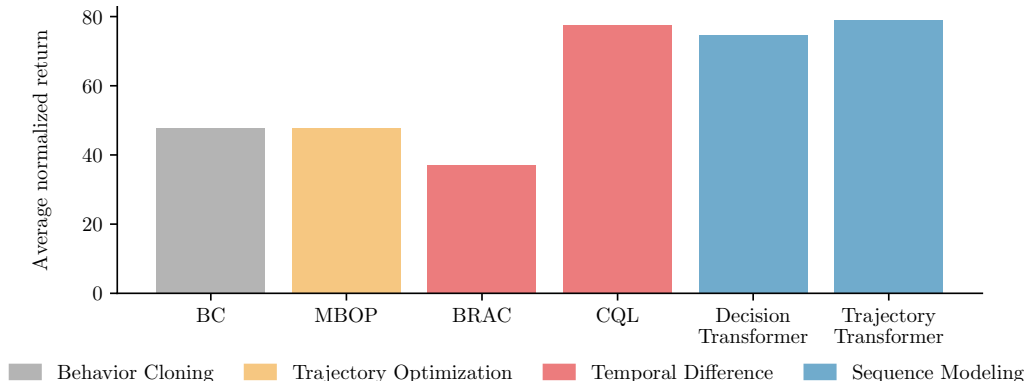
Figure 4.5: **(Offline locomotion performance)** A plot showing the average per-algorithm performance in Table 4.1, with bars colored according to a crude algorithm categorization. In this plot, "Trajectory Transformer" refers to the quantile discreization variant.

similarly on all environments except for HalfCheetah-Medium-Expert, where the large range of the velocities prevents the uniform discretization scheme from recovering the precise actuation required for enacting the expert policy. As a result, the quantile discretization approach achieves a return of more than twice that of the uniform discretization.

Combining with Q-functions. Though Monte Carlo value estimates are sufficient for many standard offline reinforcement learning benchmarks, in sparse-reward and long-horizon settings they become too uninformative to guide the beam-search-based planning procedure. In these problems, the value estimate from the Transformer can be replaced with a Q-function trained via dynamic programming. We explore this combination by using the Q-function from the implicit Q-learning algorithm (IQL; Kostrikov et al. 2022) on the AntMaze navigation tasks (Fu et al., 2020), for which there is only a sparse reward upon reaching the goal state. These tasks evaluate temporal compositionality because they require stitching together multiple zero-reward trajectories in the dataset to reach a designated goal.

AntMaze results are provided in Table 4.2. Q-guided Trajectory Transformer planning outperforms all prior methods on all maze sizes and dataset compositions. In particular, it outperforms the IQL method from which we obtain the Q-function, underscoring that planning with a Q-function as a search heuristic can be less susceptible to errors in the Q-function than policy extraction. However, because the Q-guided planning procedure still benefits from the temporal compositionality of both dynamic programming and planning, it outperforms return-conditioning approaches, such as the Decision Transformer, that suffer due to the lack of complete demonstrations in the AntMaze datasets.

---

[1]More attention visualizations can be found at `trajectory-transformer.github.io/attention`

Dataset	Environment	BC	CQL	IQL	DT	TT $(+Q)$
Umaze	AntMaze	54.6	74.0	87.5	59.2	100.0 ±0.0
Medium-Play	AntMaze	0.0	61.2	71.2	0.0	93.3 ±6.4
Medium-Diverse	AntMaze	0.0	53.7	70.0	0.0	100.0 ±0.0
Large-Play	AntMaze	0.0	15.8	39.6	0.0	66.7 ±12.2
Large-Diverse	AntMaze	0.0	14.9	47.5	0.0	60.0 ±12.7
Average		10.9	44.9	63.2	11.8	84.0

Table 4.2: **(Combining with Q-functions)** Performance on the sparse-reward AntMaze (v0) navigation task. Using a Q-function as a search heuristic with the Trajectory Transformer (TT $(+Q)$) outperforms policy extraction from the Q-function (IQL) and return-conditioning approaches like the Decision Transformer (DT). We report means and standard error over 15 random seeds for TT $(+Q)$; baseline results are taken from (Kostrikov et al., 2022).

Imitation and goal-reaching. We additionally plan with the Trajectory Transformer using standard likelihood-maximizing beam search, as opposed to the return-maximizing version used for offline RL. We find that after training the model on datasets collected by expert policies (Fu et al., 2020), using beam search as a receding-horizon controller achieves an average normalized return of 104% and 109% in the Hopper and Walker2d environments, respectively, using the same evaluation protocol of 15 runs described as in the offline reinforcement learning results. While this result is perhaps unsurprising, as behavior cloning with standard feedforward architectures is already able to reproduce the behavior of the expert policies, it demonstrates that a decoding algorithm used for language modeling can be effectively repurposed for control.

Finally, we evaluate the goal-reaching variant of beam-search, which conditions on a future desired state alongside previously encountered states. We use a continuous variant of the classic four rooms environment as a testbed (Sutton et al., 1999). Our training data consists of trajectories collected by a pretrained goal-reaching agent, with start and goal states sampled uniformly at random across the state space. Figure 4.6 depicts routes taken by the the planner. Anti-causal conditioning on a future state allows for beam search to be used as a goal-reaching method. No reward shaping, or rewards of any sort, are required; the planning method relies entirely on goal relabeling. An extension of this experiment to procedurally-generated maps is described in Appendix B.6.
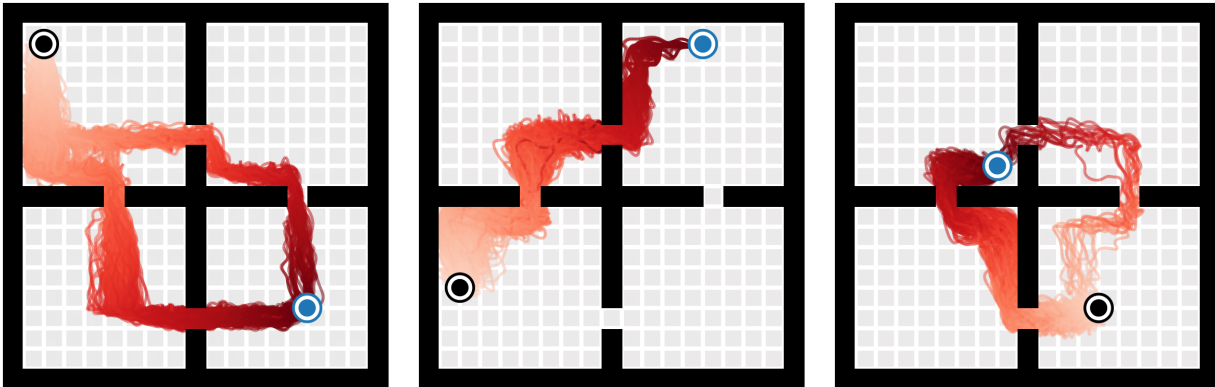
Figure 4.6: **(Goal-reaching)** Trajectories collected by TTO with anti-causal goal-state conditioning in a continuous variant of the four rooms environment. Trajectories are visualized as curves passing through all encountered states, with color becoming more saturated as time progresses. Note that these curves depict real trajectories collected by the controller and not sampled sequences. The starting state is depicted by ◉ and the goal state by ◉. Best viewed in color.

4.5 Discussion

We have presented a sequence modeling view on reinforcement learning that enables us to derive a single algorithm for a diverse range of problem settings, unifying many of the standard components of reinforcement learning algorithms (such as policies, models, and value functions) under a single sequence model. The algorithm involves training a sequence model jointly on states, actions, and rewards and sampling from it using a minimally modified beam search. Despite drawing from the tools of large-scale language modeling instead of those normally associated with control, we find that this approach is effective in imitation learning, goal-reaching, and offline reinforcement learning.

However, prediction with Transformers is currently slower and more resource-intensive than prediction with the types of single-step models often used in model-based control, requiring up to multiple seconds for action selection when the context window grows too large. This precludes real-time control with standard Transformers for most dynamical systems. While the beam-search-based planner is conceptually an instance of model-predictive control, and as such could be applicable wherever model-based RL is, in practice the slow planning also makes online RL experiments unwieldy. (Computationally-efficient Transformer architectures (Tay et al., 2021) could potentially cut runtimes down substantially.) Further, we have chosen to discretize continuous data to fit a standard architecture instead of modifying the architecture to handle continuous inputs. While we found this design to be much more effective than conventional continuous dynamics models, it does in principle impose an upper bound on

prediction precision.

The effectiveness of the Trajectory Transformer stems largely from its improved predictive accuracy compared to dynamics models used in prior model-based reinforcement learning algorithms; its weaknesses are largely consequences of the beam search-based planner in which it is embedded. In the next chapter, we will consider whether it is possible to retain the strengths of the Trajectory Transformer without suffering from its limitations by building a planning algorithm from the ground up around the affordances of a specific generative model, as opposed to using an improved generative model in a generic search algorithm for model-predictive control.

5

Planning with Diffusion

5.1 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 conventional search algorithms (Reddy, 1977; Kocsis & Szepesvári, 2006) or trajectory optimization routines (Tassa et al., 2012; Posa et al., 2014; Kelly, 2017), as demonstrated with the Trajectory Transformer in Chapter 4.

However, this combination has a number of shortcomings. Because powerful trajectory optimizers can exploit learned models, plans generated by better optimizers 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. Because these problems stem from the limitations of the planning algorithm, they cannot be fully overcome by improved model quality.

In this chapter, 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,
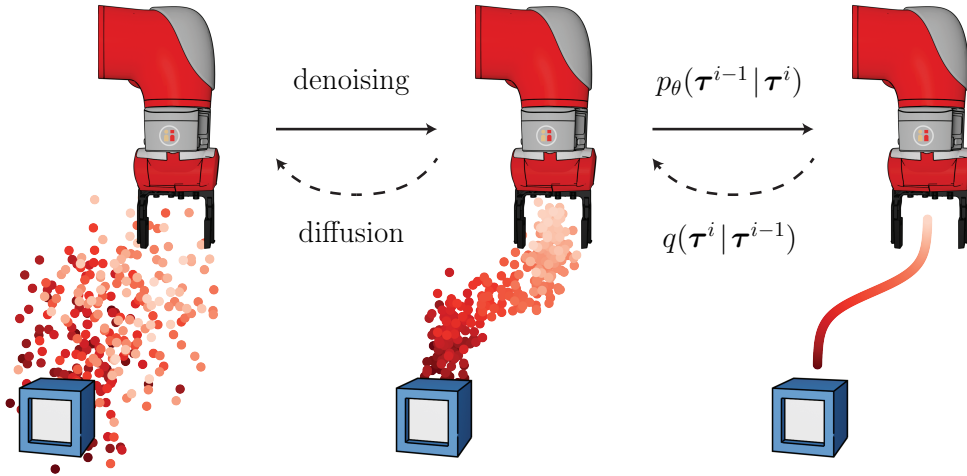
Figure 5.1: **(Planning via denoising)** Diffuser is a diffusion probabilistic model that plans by iteratively refining trajectories.

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 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 5.2. 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
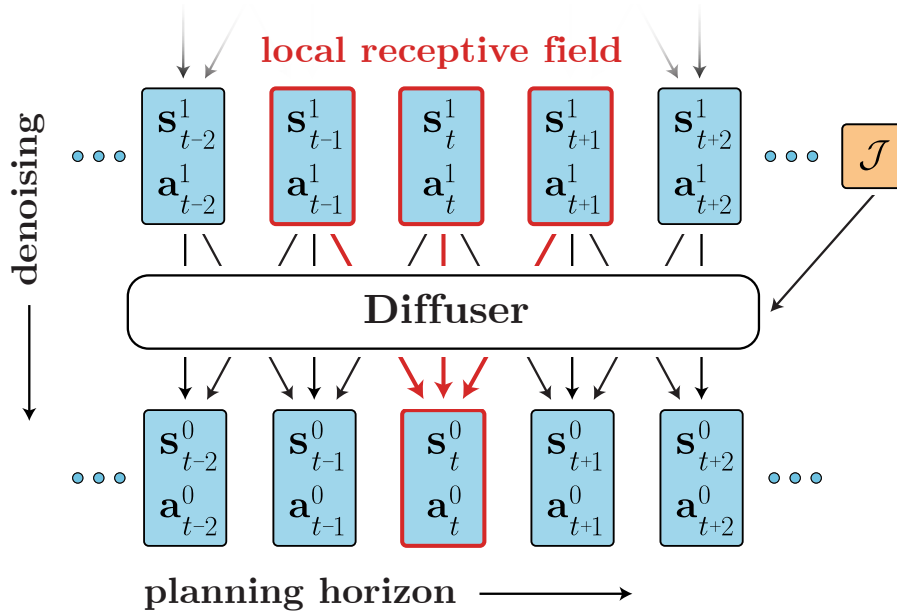
Figure 5.2: **(Diffuser block architecture)** Diffuser samples plans by iteratively denoising two-dimensional arrays consisting of a variable number of state-action pairs. A small receptive field constrains the model to only enforce local consistency during a single denoising step. By composing many 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.

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 chapter 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, the unusual properties of Diffuser make it particularly effective in control settings that require long-horizon reasoning and test-time flexibility.

5.2 Background on 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: $\theta^* = \arg\min_\theta -\mathbb{E}_{\boldsymbol{\tau}^0} [\log p_\theta(\boldsymbol{\tau}^0)]$. 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 tth 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 tth state (or action) in a trajectory $\boldsymbol{\tau}$ as $\boldsymbol{\tau}_{\mathbf{s}_t}$ (or $\boldsymbol{\tau}_{\mathbf{a}_t}$).

5.3 Planning with Diffusion

A major obstacle to using trajectory optimization techniques is that they require knowledge of the environment dynamics. 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:

$$\tilde{p}_\theta(\boldsymbol{\tau}) \propto p_\theta(\boldsymbol{\tau})h(\boldsymbol{\tau}). \tag{5.1}$$

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, 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.

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 5.2, 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

---

[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).

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:

$$\boldsymbol{\tau} = \begin{bmatrix} \mathbf{s}_0 & \mathbf{s}_1 & \cdots & \mathbf{s}_T \\ \mathbf{a}_0 & \mathbf{a}_1 & & \mathbf{a}_T \end{bmatrix}. \tag{5.2}$$

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, 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 C.1). 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 $i \sim \mathcal{U}\{1, 2, \ldots, N\}$ 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).

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 5.1:

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

Algorithm 5 Guided Diffusion Planning

1: **Require** Diffuser μ_θ, guide $\mathcal{J}$, scale α, covariances Σ^i
2: **while** not done **do**
3:     Observe state **s**; initialize plan $\boldsymbol{\tau}^N \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
4:     **for** $i = N, \ldots, 1$ **do**
5:         `// parameters of reverse transition`
6:         $\mu \leftarrow \mu_\theta(\boldsymbol{\tau}^i)$
7:         `// guide using gradients of return`
8:         $\boldsymbol{\tau}^{i-1} \sim \mathcal{N}(\mu + \alpha\Sigma\nabla\mathcal{J}(\mu), \Sigma^i)$
9:         `// constrain first state of plan`
10:        $\boldsymbol{\tau}^{i-1}_{\boldsymbol{s}_0} \leftarrow \boldsymbol{s}$
11:     **end for**
12:     Execute first action of plan $\boldsymbol{\tau}^0_{\mathbf{a}_0}$
13: **end while**

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):

$$p_\theta(\boldsymbol{\tau}^{i-1} \mid \boldsymbol{\tau}^i, \mathcal{O}_{1:T}) \approx \mathcal{N}(\boldsymbol{\tau}^{i-1}; \mu + \Sigma g, \Sigma) \tag{5.3}$$

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

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

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 5.3. 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 5.

Goal-Conditioned Reinforcement Learning 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 5.2, 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

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

The definition for action constraints is identical. In practice, this may be implemented by sampling from 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 $i \in \{0, 1, \ldots, N\}$.

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 5.

5.4 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 $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$, and as such are not tied to any planning algorithm in particular. In contrast, the planning routine in Algorithm 5 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 5.3a**, showing that Diffuser is able to generate feasible trajectories in the types of
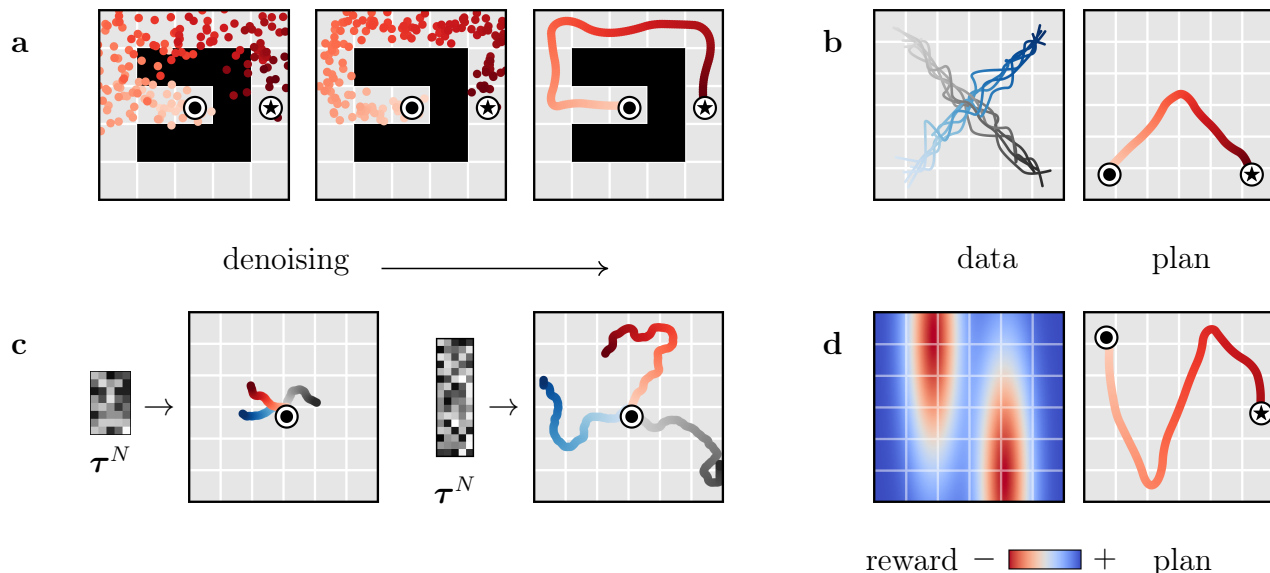
Figure 5.3: **(Properties of diffusion planners) (a) Learned long-horizon planning:** Diffuser's learned planning procedure does not suffer from the myopic failure modes common to shooting algorithms and is able to plan over long horizons with sparse reward. **(b) Temporal compositionality:** Even though 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.

sparse reward settings where shooting-based approaches are known to struggle. We explore a more quantitative version of this problem setting in Section 5.5.

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 5.3), it can also stitch together familiar subsequences in novel ways. In **Figure 5.3b**, 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

Environment		MPPI	CQL	IQL	Diffuser
Maze2D	U-Maze	33.2	5.7	47.4	**113.9** ±3.1
Maze2D	Medium	10.2	5.0	34.9	**121.5** ±2.7
Maze2D	Large	5.1	12.5	58.6	**123.0** ±6.4
Single-task Average		16.2	7.7	47.0	**119.5**
Multi2D	U-Maze	41.2	-	24.8	**128.9** ±1.8
Multi2D	Medium	15.4	-	12.1	**127.2** ±3.4
Multi2D	Large	8.0	-	13.9	**132.1** ±5.8
Multi-task Average		21.5	-	16.9	**129.4**

Table 5.1: **(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 C.1 details the sources for the scores of the baseline algorithms.

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 5.3c**).

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 5.3d**).

5.5   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.

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 5.1).

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 5.4.

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 5.1. 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 C.1. MPPI uses the ground-truth dynamics; its poor performance compared to the learned planning algorithm of Diffuser highlights the difficulty posed by long-horizon planning even when there are no prediction inaccuracies.

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.
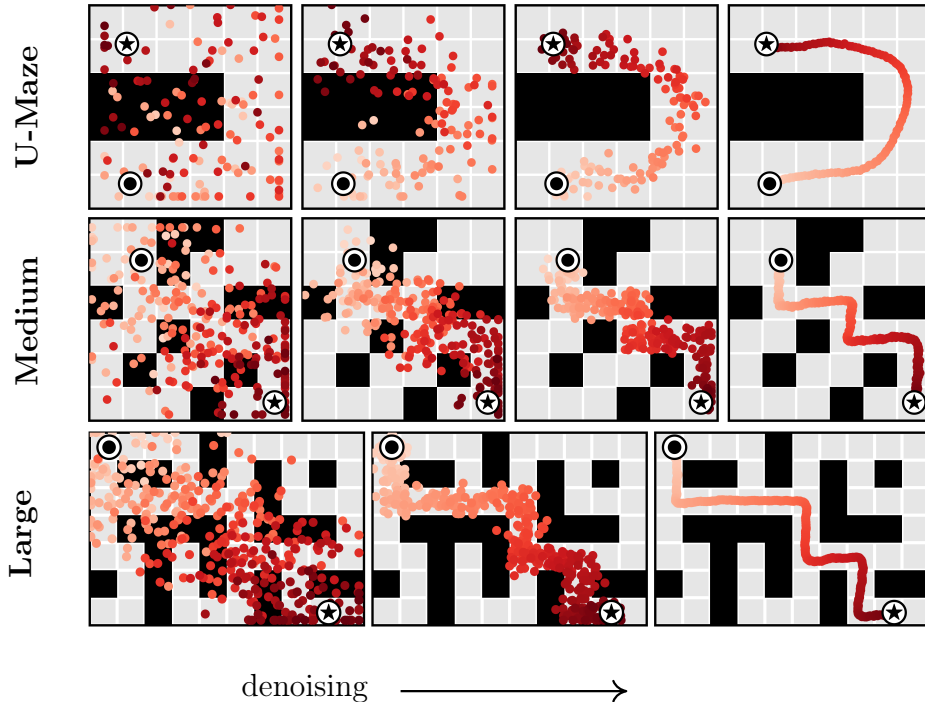
denoising $\longrightarrow$

Figure 5.4: **(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.

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	8.1	**54.4**

Table 5.2: **(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.

We use one trained Diffuser for all block-stacking tasks, only 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 C.2 for details.)
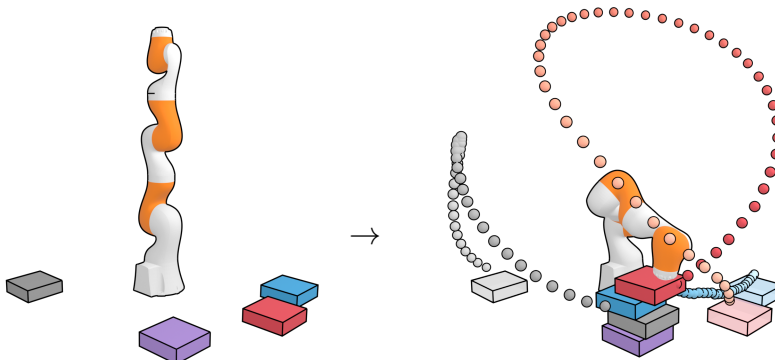
Figure 5.5: **(Block stacking)** A block stacking sequence executed by Diffuser. This task is best illustrated by videos viewable at `diffusion-planning.github.io`.

We compare with two prior model-free offline reinforcement learning algorithms: BCQ (Fujimoto et al., 2019) and CQL (Kumar et al., 2020a), training standard variants for the unconditional stacking task and goal-conditioned variants for the conditional stacking and rearrangement tasks. (Baseline details are provided in Appendix C.1.) Quantitative results are given in Table 5.2, 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.

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 Algorithm 5 and condition the trajectories on the current state using the inpainting procedure described in Section 5.3. The reward predictor $\mathcal{J}_\phi$ is trained on the same trajectories as the diffusion model. A visualization of the learned denoising procedure in this setting is shown in Figure 5.6.

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., 2020a) and IQL (Kostrikov et al., 2022); 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), MOPO (Yu et al., 2020), MOReL (Kidambi et al., 2020), and MBOP (Argenson & Dulac-Arnold, 2021). As shown in Table 5.3, Diffuser performs comparably to prior algorithms in the single-task setting: better than the model-based
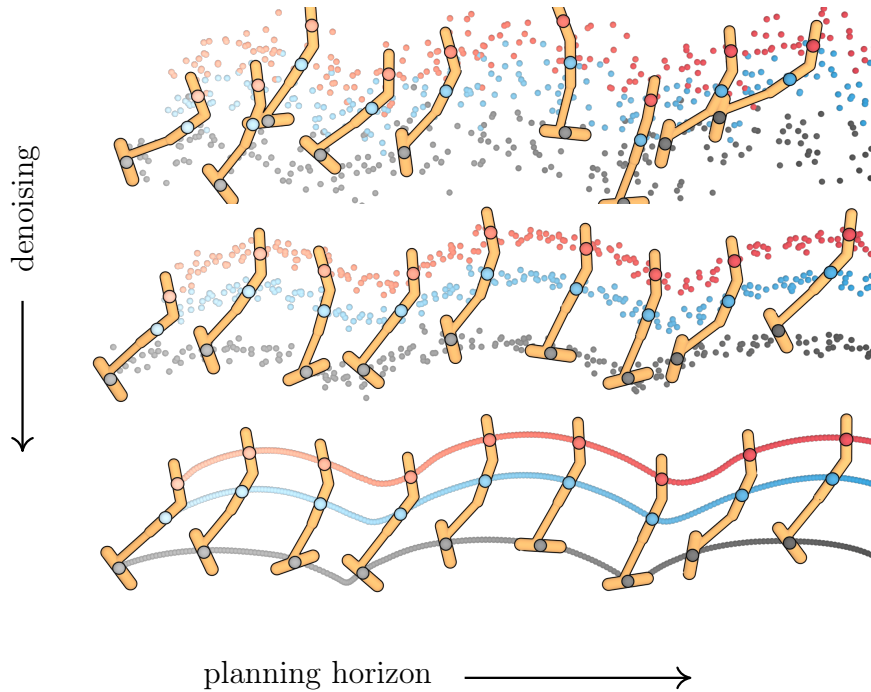
Figure 5.6: **(Guided sampling)** Diffuser generates all timesteps of a plan concurrently, instead of autoregressively, through the denoising process.

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 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.

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 5.7, we illustrate the trade-off between performance and runtime budget as we vary the underlying number

Dataset	Environment	BC	CQL	IQL	DT	TT	MOPO	MBOP	Diffuser
Medium-Expert	HalfCheetah	55.2	91.6	86.7	86.8	95.0	63.3	**105.9**	88.9 ±0.3
Medium-Expert	Hopper	52.5	**105.4**	91.5	**107.6**	**110.0**	23.7	55.1	103.3 ±1.3
Medium-Expert	Walker2d	**107.5**	**108.8**	**109.6**	**108.1**	101.9	44.6	70.2	**106.9** ±0.2
Medium	HalfCheetah	42.6	44.0	**47.4**	42.6	**46.9**	42.3	44.6	42.8 ±0.3
Medium	Hopper	52.9	58.5	66.3	67.6	61.1	28.0	48.8	**74.3** ±1.4
Medium	Walker2d	75.3	72.5	**78.3**	74.0	**79.0**	17.8	41.0	**79.6** ±0.55
Medium-Replay	HalfCheetah	36.6	45.5	44.2	36.6	41.9	**53.1**	42.3	37.7 ±0.5
Medium-Replay	Hopper	18.1	**95.0**	**94.7**	82.7	**91.5**	67.5	12.4	**93.6** ±0.4
Medium-Replay	Walker2d	26.0	77.2	73.9	66.6	**82.6**	39.0	9.7	70.6 ±1.6
Average		51.9	**77.6**	**77.0**	74.7	**78.9**	42.1	47.8	**77.5**

Table 5.3: **(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 150 planning seeds. We detail the sources for the performance of prior methods in Appendix C.1. Following Kostrikov et al. (2022), we emphasize in bold scores within 5 percent of the maximum per task ($\geq 0.95 \cdot \max$).

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.

5.6 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). Further, Lambert et al. 2020 have studied non-autoregressive trajectory-level dynamics models for long-horizon prediction. 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

Performance versus runtime
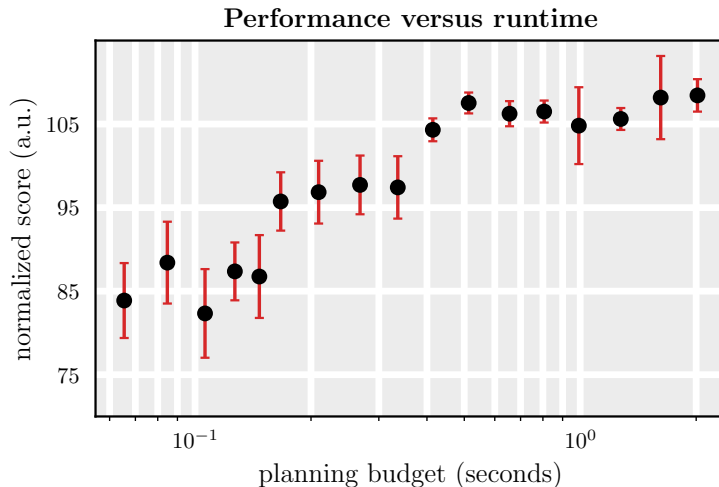
Figure 5.7: **(Warm-starting planning)** Performance of Diffuser on Walker2d Medium-Expert when varying the number of diffusion steps to warm-start planning. Performance suffers only minimally even when using one-tenth the number of diffusion steps, as long as plans are initialized from the previous timestep's plan.

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 (Du et al., 2019; Rybkin et al., 2021). In contrast, our method plans by modeling and 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, 2019a), 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.

5.7   Discussion

This chapter has proposed Diffuser, a denoising diffusion model for trajectory data. Unlike conventional dynamics models, Diffuser lends itself to a learned, non-greedy planning procedure that allows it so solve long-horizon tasks. 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. However, there are genuine tradeoffs between Diffuser and the Transformer-based approach from Chapter 4. While Diffuser comes equipped with a much more efficient learned planner than beam search, it comes at the cost of reduced open-loop predictive accuracy. This discrepancy arises due to differences in the underlying generative models. Diffusion models' success in image generation has been attributed to an increased importance placed on low-frequency information, a consequence of the denoising schedule and parameter-sharing across noise levels (Song & Ermon, 2019b; Dieleman, 2022). In contrast, Transformers are more readily able to represent the types of high-frequency information that can disproportionately affect trajectories, such as joint angles near a contact point, but are limited in planning contexts due to impoverished search procedures. It remains an open question, both in the context of decision-making and broadly in generative modeling, how to best combine the effectiveness of the Transformer architecture with the strengths of non-autoregressive denoising strategies.

6

Conclusion

In this thesis, we considered the role of generative models in the reinforcement learning problem. We began by asking what prediction problem the generative model should be trained to solve, and in the process developed a continuous-state dynamics model that can predict over probabilistic, infinite horizons. However, when generalizing the standard single-step prediction problem to that of infinite-horizon prediction, we found that the bottleneck to scaling lay with the quality of the generative model itself. As a result, we studied the extent to which improved generative modeling capabilities could be used to enhance the performance of a planning algorithm, designing a planner around the toolbox of large-scale language modeling. While this approach markedly increased predictive accuracy, planning capabilities lagged behind due to the inefficiency of language model decoding approaches, making the new bottleneck the planning algorithm in which the generative model was embedded.

The main difficulty with such an approach was that the quality of the planner was constant; because the model itself could improve with data but the planner could not, the model would continue to become more accurate until the planner became the performance-limiting component. To address this, we described a method of incorporating the entire planning algorithm into the generative model itself, as opposed to embedding a generative model into a conventional planner. This design allowed the plans, and not just the predictive quality, to improve with more data and experience, and we demonstrated that this approach was effective in long-horizon and sparse-reward settings where conventional planning algorithms struggle.

As with many research programs, the work in this thesis raises more questions than it answers. We conclude by discussing some of the most promising directions for future work in the intersection of reinforcement learning and generative modeling:

- **Planning in abstracted time.** The γ-model allows us to ask questions of the form "will I reach a particular state at some point in the future" without needing to worry

about precisely when that state will be reached. This affordance is a natural fit for long-horizon decision-making. For example, a plan for a mobile robot would plausibly need to include battery recharging after some amount of time. The ability to incorporate such a step in its plan, without needing to precommit to the exact second in which this step must occur, would allow the planner to hone in on higher-level decisions to be made and their ordering as opposed to unnecessary (and inevitably misleading) precision. This idea shares much of its inspiration with the line of work describing options (Sutton et al., 1999); scaling such methods to high-dimensional control problems poses an exciting, and likely fruitful, challenge.

- **Planning in abstracted states.** The methods described in this thesis plan in raw state space. In large observation spaces, this becomes not only unnecessary – as most details in an observation are unlikely to be relevant to the task at hand – but also possibly infeasible in the event that the space is so large that high-quality generative modeling once again becomes the bottleneck. Instead, it would be much more efficient to plan in a compressed space pruned of unnecessary details. There has been much work on learned latent spaces, with the usual spectrum ranging from those trained with reconstructive objectives (Hafner et al., 2021a; Janner et al., 2019) to those that contain only value-relevant information (Grimm et al., 2020) and options that interpolate between these two extremes (Farahmand et al., 2017; Eysenbach et al., 2021). However, it remains an open problem how to combine these works with learned planners, contemporary high-capacity generative models, and deploy them in the types of real-world problems that are often used as the motivating examples for learned latent spaces.

- **Improved learned planning algorithms.** One of the main contributions of this thesis is a learned planning algorithm based on iterative denoising. The main advantage of this method compared to conventional planners is its ability to improve with experience. However, just as there exist many classical planning algorithms with different strengths and weaknesses, there is no reason to believe that Diffuser is the end of the story for data-driven planning. Improved variants could incorporate dynamics Jacobians, more sophisticated constrained optimization techniques, or make use of tree-based search techniques for better coverage of the space of possibilities.

- **Amortized and adaptive planning.** As discussed in Chapter 1, model-based planners are appealing for their test-time flexibility and generalization properties. However, a fundamental limitation of planning-based approaches is their runtime costs, which can become prohibitively large for any problem requiring a high planning frequency. In comparison, model-free policies tend to be more efficient, often requiring only a single feedforward pass of a neural network as opposed to many such computations for the purpose of an iterative planner. To benefit from the respective strengths of both approaches, the cost of planning could be amortized over time by storing the result of the planning method into a lightweight policy and falling back on the planner only in

novel situations. The open challenge, of course, is to design a reliable mechanism for determining when a situation calls for the slower planning approach and when it is safe to use the faster policy.

- **Foundation dynamics models.** The dynamics models and planners described in this thesis were trained and evaluated on individual tasks (*e.g.*, in locomotion settings) or in small sets of tasks (*e.g.*, in maze-solving or block-stacking domains). While common in reinforcement learning, this workflow is becoming increasingly rare in the rest of machine learning, in which models pretrained on large-scale datasets are finetuned (or used directly) on a wide variety of tasks. There is an opportunity to build something similar for dynamics models, especially in real-world robotics settings where the regularity of the physical world could enable the types of generalization present in large language modeling (Brown et al., 2020). In the short term, this problem is largely one of dataset collection. In the longer term, this endeavour will open up new research problems we are not yet able to anticipate. A common refrain in deep learning is that something new breaks with every order-of-magnitude scale increase, and it is likely that the same will be true in robotic control.

Open-Source Implementations

Code to reproduce the results in this thesis is available at the following webpages:

- Chapter 3: `gamma-models.github.io`

- Chapter 4: `trajectory-transformer.github.io`

- Chapter 5: `diffusion-planning.github.io`

Code References

The following libraries were used in this work: NumPy (Harris et al., 2020), PyTorch (Paszke et al., 2019), JAX (Bradbury et al., 2018), Flax (Heek et al., 2023), einops (Rogozhnikov, 2022), MuJoCo (Todorov et al., 2012), mujoco-py (OpenAI, 2016), Gym (Brockman et al., 2016), minGPT (Karpathy, 2020), and Diffusion Models in PyTorch (Wang, 2020).

Bibliography

Amos, B., Rodriguez, I. D. J., Sacks, J., Boots, B., and Kolter, J. Z. Differentiable mpc for end-to-end planning and control. In *Advances in Neural Information Processing Systems*, 2018.

Amos, B., Stanton, S., Yarats, D., and Wilson, A. G. On the model-based stochastic value gradient for continuous reinforcement learning. In *Conference on Learning for Dynamics and Control*, 2021.

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.

Asadi, K., Misra, D., Kim, S., and Littman, M. L. Combating the compounding-error problem with a multi-step model. *arXiv preprint arXiv:1905.13320*, 2019.

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.

Bakker, B. Reinforcement learning with long short-term memory. In *Neural Information Processing Systems*, 2002.

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.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems 30*. 2017.

Barreto, A., Borsa, D., Quan, J., Schaul, T., Silver, D., Hessel, M., Mankowitz, D., Zidek, A., and Munos, R. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *Proceedings of the International Conference on Machine Learning*, 2018.

Bellman, R. *Dynamic Programming*. Dover Publications, 1957.

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.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, 2018a.

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, 2018b.

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.

Chiappa, S., Racaniere, S., Wierstra, D., and Mohamed, S. Recurrent environment simulators. In *International Conference on Learning Representations*, 2017.

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.

Co-Reyes, J., Liu, Y., Gupta, A., Eysenbach, B., Abbeel, P., and Levine, S. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *International Conference on Machine Learning*, 2018.

Dadashi, R., Rezaeifar, S., Vieillard, N., Hussenot, L., Pietquin, O., and Geist, M. Offline reinforcement learning with pseudometric learning. *arXiv preprint arXiv:2103.01948*, 2021.

Dayan, P. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5:613, 1993.

Deisenroth, M. and Rasmussen, C. E. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*, 2011.

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

Diehl, M., Ferreau, H. J., and Haverbeke, N. *Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation*. 2009.

Dieleman, S. Diffusion models are autoencoders, 2022. URL https://benanne.github.io/2022/01/31/diffusion.html.

Dong, K., Luo, Y., Yu, T., Finn, C., and Ma, T. On the expressivity of neural networks for deep reinforcement learning. In *International Conference on Machine Learning*, 2020.

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.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Neural spline flows. In *Advances in Neural Information Processing Systems*. 2019.

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.

Bibliography

Fairbank, M. Reinforcement learning by value gradients. *arXiv preprint arXiv:0803.3539*, 2008.

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.

Farshidian, F., Neunert, M., and Buchli, J. Learning of closed-loop motion control. In *International Conference on Intelligent Robots and Systems*, 2014.

Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E., and Levine, S. Model-based value estimation for efficient model-free reinforcement learning. In *International Conference on Machine Learning*, 2018.

Foster, D. and Dayan, P. Structure in the space of value functions. *Machine Learning*, 49: 325, 2002.

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.

Gershman, S. J. The successor representation: Its computational logic and neural substrates. *Journal of Neuroscience*, 2018.

Ghasemipour, S. K. S., Schuurmans, D., and Gu, S. S. EMaQ: Expected-max Q-learning operator for simple yet effective offline and online rl. *arXiv preprint arXiv:2007.11091*, 2020.

Ghosh, D., Gupta, A., Reddy, A., Fu, J., Devin, C. M., Eysenbach, B., and Levine, S. Learning to reach goals via iterated supervised learning. In *International Conference on Learning Representations*, 2021.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014a.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014b.

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.

Grimm, C., Barreto, A., Singh, S., and Silver, D. The value equivalence principle for model-based reinforcement learning. In *Neural Information Processing Systems*, 2020.

Gu, J., Liu, Q., and Cho, K. Insertion-based Decoding with Automatically Inferred Generation Order. *Transactions of the Association for Computational Linguistics*, 2019.

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

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 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.

Hansen, S., Dabney, W., Barreto, A., Warde-Farley, D., de Wiele, T. V., and Mnih, V. Fast task inference with variational intrinsic successor features. In *International Conference on Learning Representations*, 2020.

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.

Heek, J., Levskaya, A., Oliver, A., Ritter, M., Rondepierre, B., Steiner, A., and van Zee, M. Flax: A neural network library and ecosystem for JAX, 2023. URL http://github.com/google/flax.

Heess, N., Hunt, J. J., Lillicrap, T., and Silver, D. Memory-based control with recurrent neural networks. In *Neural Information Processing Systems Deep Reinforcement Learning Workshop*, 2015a.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Tassa, Y., and Erez, T. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, 2015b.

Bibliography

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

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

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

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, 2019.

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.

Jiang, Y., Gu, S., Murphy, K., and Finn, C. Language as an abstraction for hierarchical deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2019.

Jin, Y., Yang, Z., and Wang, Z. Is pessimism provably efficient for offline RL? In *International Conference on Machine Learning*, 2021.

Jordan, M. I. and Rumelhart, D. E. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16:307, 1992.

Kaelbling, L. P. Learning to achieve goals. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1993.

Kaelbling, L. P. and Lozano-Pérez, T. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, 2011.

Kahn, G., Villaflor, A., Ding, B., Abbeel, P., and Levine, S. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *International Conference on Robotics and Automation*, 2018.

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.

Kalweit, G. and Boedecker, J. Uncertainty-driven imagination for continuous deep reinforcement learning. In *Conference on Robot Learning*, 2017.

Kaplan, J., McCandlish, S., Henighan, T. J., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Karpathy, A. minGPT: A minimal pytorch re-implementation of the openai gpt training, 2020. URL https://github.com/karpathy/minGPT.

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.

Kocsis, L. and Szepesvári, C. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, 2006.

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

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems*. 2012.

Kulkarni, T. D., Saeedi, A., Gautam, S., and Gershman, S. J. Deep successor reinforcement learning, 2016.

Kumar, A., Fu, J., Tucker, G., and Levine, S. Stabilizing off-policy Q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, 2019a.

Kumar, A., Peng, X. B., and Levine, S. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*, 2019b.

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

Kumar, A., Agarwal, R., Ma, T., Courville, A., Tucker, G., and Levine, S. DR3: Value-based deep reinforcement learning requires explicit regularization. In *International Conference on Learning Representations*, 2022.

Kumar, S., Parker, J., and Naderian, P. Adaptive transformers in RL. *arXiv preprint arXiv:2004.03761*, 2020b.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. Model-ensemble trust-region policy optimization. In *International Conference on Learning Representations*, 2018.

Lambert, N. O., Wilcox, A., Zhang, H., Pister, K. S., and Calandra, R. Learning accurate long-term dynamics for model-based reinforcement learning. *arXiv preprint arXiv:2012.09156*, 2020.

Lampe, T. and Riedmiller, M. Approximate model-assisted neural fitted Q-iteration. In *International Joint Conference on Neural Networks*, 2014.

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.

Levine, S. and Koltun, V. Guided policy search. In *International Conference on Machine Learning*, 2013.

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

Luo, Y., Xu, H., Li, Y., Tian, Y., Darrell, T., and Ma, T. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. In *International Conference on Learning Representations*, 2019.

Ma, C., Wen, J., and Bengio, Y. Universal successor representations for transfer reinforcement learning. *arXiv preprint arXiv:1804.03758*, 2018.

Malik, A., Kuleshov, V., Song, J., Nemer, D., Seymour, H., and Ermon, S. Calibrated model-based deep reinforcement learning. In *International Conference on Machine Learning*, 2019.

Mao, X., Li, Q., Xie, H., Lau, R. Y. K., and Wang, Z. Least squares generative adversarial networks. *arXiv preprint arXiv:1611.04076*, 2016.

Meister, C., Cotterell, R., and Vieira, T. If beam search is the answer, what was the question? In *Empirical Methods in Natural Language Processing*, 2020.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 2015.

Momennejad, I., Russek, E. M., Cheong, J. H., Botvinick, M. M., Daw, N. D., and Gershman, S. J. The successor representation in human reinforcement learning. *Nature Human Behaviour*, 1(9):680–692, 2017.

Moore, A. W. *Efficient Memory-based Learning for Robot Control*. PhD thesis, University of Cambridge, 1990.

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.

Nagabandi, A., Konoglie, K., Levine, S., and Kumar, V. Deep Dynamics Models for Learning Dexterous Manipulation. In *Conference on Robot Learning*, 2019.

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

Nguyen, D. H. and Widrow, B. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, 1990.

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.

Nowozin, S., Cseke, B., and Tomioka, R. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems*. 2016.

Oh, J., Chockalingam, V., Lee, H., et al. Control of memory, active perception, and action in Minecraft. In *International Conference on Machine Learning*, 2016.

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

OpenAI. mujoco-py, 2016. URL https://github.com/openai/mujoco-py.

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.

Parisotto, E. and Salakhutdinov, R. Efficient transformers in reinforcement learning using actor-learner distillation. In *International Conference on Learning Representations*, 2021.

Parisotto, E., Song, F., Rae, J., Pascanu, R., Gulcehre, C., Jayakumar, S., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, 2020.

Paster, K., McIlraith, S. A., and Ba, J. Planning from pixels using inverse dynamics models. In *International Conference on Learning Representations*, 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.

Peng, X. B., Berseth, G., Yin, K., and Van De Panne, M. DeepLoco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics*, 2017.

Pong, V., Gu, S., Dalal, M., and Levine, S. Temporal difference models: Model-free deep RL for model-based control. In *International Conference on Learning Representations*, 2018.

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.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pre-training. 2018.

Rauber, P., Ummadisingu, A., Mutz, F., and Schmidhuber, J. Hindsight policy gradients. In *International Conference on Learning Representations*, 2019.

Reddy, R. Speech understanding systems: Summary of results of the five-year research effort at Carnegie Mellon University, 1977.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *Proceedings of Machine Learning Research*, 2015.

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

Rogozhnikov, A. Einops: Clear and reliable tensor manipulations with einstein-like notation. In *International Conference on Learning Representations*, 2022.

Ross, S. and Bagnell, D. Efficient reductions for imitation learning. In *International Conference on Artificial Intelligence and Statistics*, 2010.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, 2011.

Rybkin, O., Zhu, C., Nagabandi, A., Daniilidis, K., Mordatch, I., and Levine, S. Model-based reinforcement learning via latent-space collocation. In *International Conference on Machine Learning*, pp. 9190–9201. PMLR, 2021.

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.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *Proceedings of the International Conference on Machine Learning*, 2015.

Schmidhuber, J. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

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.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Silver, D., Sutton, R. S., and Müller, M. Sample-based learning and search with permanent and transient memories. In *International Conference on Machine Learning*, 2008.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., and Degris, T. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, 2017.

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*, 2019a.

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

Srivastava, R. K., Shyam, P., Mutz, F., Jaśkowski, W., and Schmidhuber, J. Training agents using upside-down reinforcement learning. *arXiv preprint arXiv:1912.02877*, 2019.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, 2014.

Sutton, R. TD models: Modeling the world at a mixture of time scales. In *Proceedings of the 12th International Conference on Machine Learning*, 1995.

Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine Learning*, 1988.

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

Sutton, R. S. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*. 1996.

Sutton, R. S., Precup, D., and Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 1999.

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.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. Long range arena: A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021.

Todorov, E., Erez, T., and Tassa, Y. MuJoCo: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, 2012.

van Hasselt, H. P., Hessel, M., and Aslanides, J. When to use parametric models in reinforcement learning? In *Neural Information Processing Systems*, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

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

Wang, T. and Ba, J. Exploring model-based planning with policy networks. In *International Conference on Learning Representations*, 2020.

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.

Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1989.

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.

Wu, Y., Tucker, G., and Nachum, O. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Yin, M., Bai, Y., and Wang, Y.-X. Near-optimal offline reinforcement learning via double variance reduction. *arXiv preprint arXiv:2102.01748*, 2021.

Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J., Levine, S., Finn, C., and Ma, T. MOPO: Model-based offline policy optimization. In *Advances in Neural Information Processing Systems*, 2020.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

Zhang, M. R., Paine, T., Nachum, O., Paduraru, C., Tucker, G., ziyu wang, and Norouzi, M. Autoregressive dynamics models for offline policy evaluation and optimization. In *International Conference on Learning Representations*, 2021.

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

γ-Model Details

A.1 Geometric weighting lemma

Lemma 1. *Let α_n be importance weights as described in Theorem 1:* $\alpha_n = \frac{(1-\tilde{\gamma})(\tilde{\gamma}-\gamma)^{n-1}}{(1-\gamma)^n}$. *Then:*

$$1 - \sum_{n=1}^{H} \alpha_n = \left(\frac{\tilde{\gamma} - \gamma}{1 - \gamma} \right)^H$$

Proof.

$$
\begin{aligned}
1 - \sum_{n=1}^{H} \alpha_n &= 1 - \left(\frac{1 - \tilde{\gamma}}{\tilde{\gamma} - \gamma} \right) \sum_{n=1}^{H} \left(\frac{\tilde{\gamma} - \gamma}{1 - \gamma} \right)^n \\
&= 1 - \left(\frac{1 - \tilde{\gamma}}{\tilde{\gamma} - \gamma} \right) \frac{\left(\frac{\tilde{\gamma}-\gamma}{1-\gamma} \right) - \left(\frac{\tilde{\gamma}-\gamma}{1-\gamma} \right)^{H+1}}{\frac{1-\tilde{\gamma}}{1-\gamma}} \\
&= 1 - \left(\frac{1 - \gamma}{\tilde{\gamma} - \gamma} \right) \left(\left(\frac{\tilde{\gamma} - \gamma}{1 - \gamma} \right) - \left(\frac{\tilde{\gamma} - \gamma}{1 - \gamma} \right)^{H+1} \right) \\
&= \left(\frac{\tilde{\gamma} - \gamma}{1 - \gamma} \right)^H
\end{aligned}
$$

$\square$

Table A.1: GAN γ-model hyperparameters (Algorithm 2).

Parameter	Value
Batch size	128
Number of $\mathbf{s}_e$ samples per $(\mathbf{s}_t, \mathbf{a}_t)$ pair	512
Delay parameter τ	$5 \cdot 10^{-3}$
Step size λ	$1 \cdot 10^{-4}$
Replay buffer size (off-policy prediction experiments)	$2 \cdot 10^{5}$

A.2 Implementation Details

γ-MVE algorithmic description. The γ-MVE estimator may be used for value estimation in any actor-critic algorithm. We describe the variant used in our control experiments, in which it is used in the soft actor critic algorithm (SAC; Haarnoja et al. 2018), in Algorithm 1. The γ-model update is unique to γ-MVE; the objectives for the value function and policy are identical to those in SAC. The objective for the Q-function differs only by replacing $V(\mathbf{s}_{t+1})$ with $V_{\gamma-\mathrm{MVE}}(\mathbf{s}_{t+1})$. For a detailed description of how the gradients of these objectives may be estimated, and for hyperparameters related to the training of the Q-function, value function, and policy, we refer to (Haarnoja et al., 2018).

Network architectures. For all GAN experiments, the γ-model generator μ_θ and discriminator D_ϕ are instantiated as two-layer MLPs with hidden dimensions of 256 and leaky ReLU activations. For all normalizing flow experiments, we use a six-layer neural spline flow (Durkan et al., 2019) with 16 knots defined in the interval $[-10, 10]$. The rational-quadratic coupling transform uses a three-layer MLP with hidden dimensions of 256.

Hyperparameter settings. We include the hyperparameters used for training the GAN γ-model in Table A.1 and the flow γ-model in Table A.2.

We found the original GAN (Goodfellow et al., 2014a) and the least-squares GAN (Mao et al., 2016) formulation to be equally effective for training γ-models as GANs.

Table A.2: Flow γ-model hyperparameters (Algorithm 3)

Parameter	Value
Batch size	1024
Number of $\mathbf{s}_e$ samples per $(\mathbf{s}_t, \mathbf{a}_t)$ pair	1
Delay parameter τ	$5 \cdot 10^{-3}$
Step size λ	$1 \cdot 10^{-4}$
Replay buffer size (off-policy prediction experiments)	$2 \cdot 10^5$
Single-step Gaussian variance σ^2	$1 \cdot 10^{-2}$

A.3   Environment Details

Acrobot-v1 is a two-link system (Sutton, 1996). The goal is to swing the lower link above a threshold height. The eight-dimensional observation is given by $[\cos\theta_0, \sin\theta_0, \cos\theta_1, \sin\theta_1, \frac{\mathrm{d}}{\mathrm{d}t}\theta_0, \frac{\mathrm{d}}{\mathrm{d}t}\theta_1]$. We modify it to have a one-dimensional continuous action space instead of the standard three-dimensional discrete action space. We provide reward shaping in the form of $r_{\text{shaped}} = -\cos\theta_0 - \cos(\theta_0 + \theta_1)$.

MountainCarContinuous-v0 is a car on a track (Moore, 1990). The goal is to drive the car up a high too high to summit without built-up momentum. The two-dimmensional observation space is $[x, \frac{\mathrm{d}}{\mathrm{d}t}x]$. We provide reward shaping in the form of $r_{\text{shaped}} = x$.

Pendulum-v0 is a single-link system. The link starts in a random position and the goal is to swing it upright. The three-dimensional observation space is given by $[\cos\theta, \sin\theta, \frac{\mathrm{d}}{\mathrm{d}t}\theta]$.

Reacher-v2 is a two-link arm. The objective is to move the end effector $\mathbf{e}$ of the arm to a randomly sampled goal position $\mathbf{g}$. The 11-dimensional observation space is given by $[\cos\theta_0, \cos\theta_1, \sin\theta_0, \sin\theta_1, \mathbf{g}_x, \mathbf{g}_y, \frac{\mathrm{d}}{\mathrm{d}t}\theta_0, \frac{\mathrm{d}}{\mathrm{d}t}\theta_1, \mathbf{e}_x - \mathbf{g}_x, \mathbf{e}_y - \mathbf{g}_y, \mathbf{e}_z - \mathbf{g}_z]$.

Model-based methods often make use of shaped reward functions during model-based rollouts (Chua et al., 2018). For fair comparison, when using shaped rewards we also make the same shaping available to model-free methods.
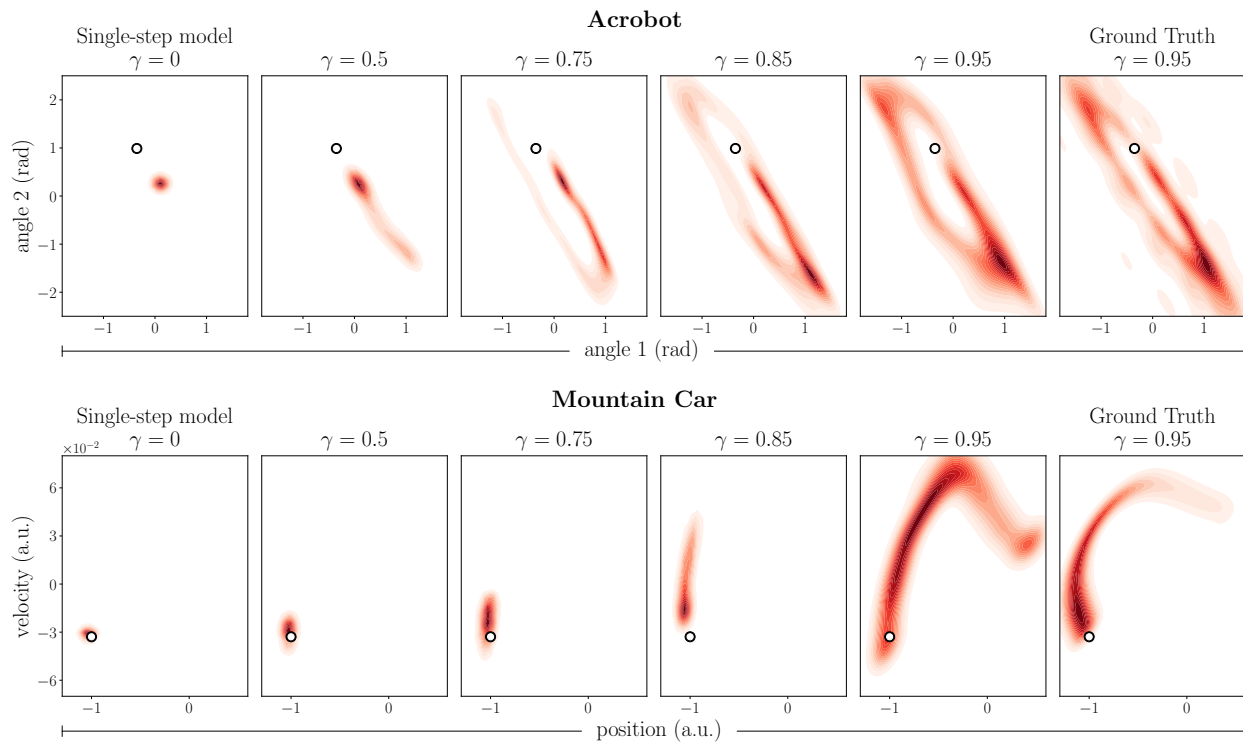
A.4 Adversarial γ-Model Predictions

Figure A.1: (**Adversarial γ-model predictions**) Visualization of the distribution from a single feedforward pass of γ-models trained as GANs according to Algorithm 2. GAN-based γ-models tend to be more unstable than normalizing flow γ-models, especially at higher discounts.

Appendix B

Trajectory Transformer Details

B.1 Model and Training Specification

Architecture and optimization details. In all environments, we use a Transformer architecture with four layers and four self-attention heads. The total input vocabulary of the model is $V \times (N + M + 2)$ to account for states, actions, rewards, and rewards-to-go, but the output linear layer produces logits only over a vocabulary of size V; output tokens can be interpreted unambiguously because their offset is uniquely determined by that of the previous input. The dimension of each token embedding is 128. Dropout is applied at the end of each block with probability 0.1.

We follow the learning rate scheduling of (Radford et al., 2018), increasing linearly from 0 to 2.5×10^{-4} over the course of 2000 updates. We use a batch size of 256.

Hardware. Model training took place on NVIDIA Tesla V100 GPUs (NCv3 instances on Microsoft Azure) for 80 epochs, taking approximately 6-12 hours (varying with dataset size) per model on one GPU.

B.2 Discrete Oracle

The discrete oracle in Figure 4.3 is the maximum log-likelihood attainable by a model under the uniform discretization granularity. For a single state dimension i, this maximum is achieved by a model that places all probability mass on the correct token, corresponding to a

uniform distribution over an interval of size

$$\frac{r_i - \ell_i}{V}.$$

The total log-likelihood over the entire state is then given by:

$$\sum_{i=1}^{N} \log \frac{V}{r_i - \ell_i}.$$

B.3 Baseline performance sources

Offline reinforcement learning The results for CQL, IQL, and DT are from Table 1 in Kostrikov et al. (2022). The results for MBOP are from Table 1 in Argenson & Dulac-Arnold (2021). The results for BRAC are from Table 2 in (Fu et al., 2020). The results for BC are from Table 1 in Kumar et al. (2020a).

B.4 Datasets

The D4RL dataset (Fu et al., 2020) used in our experiments is under the Creative Commons Attribution 4.0 License (CC BY). The license information can be found at

https://github.com/rail-berkeley/d4rl/blob/master/README.md

under the "Licenses" section.

B.5 Beam Search Hyperparameters

Beam width	maximum number of hypotheses retained during beam search	256
Planning horizon	number of transitions predicted by the model during	15
Vocabulary size	number of bins used for autoregressive discretization	100
Context size	number of input $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, R_t)$ transitions	5
$k_{\mathbf{obs}}$	top-k tokens from which observations are sampled	1
$k_{\mathbf{act}}$	top-k tokens from which actions	20

Beam width and context size are standard hyperparameters for decoding Transformer language models. Planning horizon is a standard trajectory optimization hyperparameter. The hyperparameters k_{obs} and k_{act} indicate that actions are sampled from the most likely 20% of action tokens and next observations are decoded greedily conditioned on previous observations and actions.

In many environments, the beam width and horizon may be reduced to speed up planning without affecting performance. Examples of these configurations are provided in the reference implementation: `github.com/jannerm/trajectory-transformer`.

B.6 Goal-Reaching on Procedurally-Generated Maps

The method evaluated here and the experimental setup is identical to that described in Section 3.2 (Goal-conditioned reinforcement learning), with one distinction: because the map changes each episode, the Transformer model has an additional context embedding that is a function of the current observation image. This embedding is the output of a small convolutional neural network and is added to the token embeddings analogously to the treatment of position embeddings. The agent position and goal state are not included in the map; these are provided as input tokens as described in Section 3.2.

The action space of this environment is discrete. There are seven actions, but only four are required to complete the tasks: turning left, turning right, moving forward, and opening a door. The training data is a mixture of trajectories from a pre-trained goal-reaching policy and a uniform random policy.

94% of testing goals are reached by the model on held-out maps. Example paths are shown in Figure B.1.
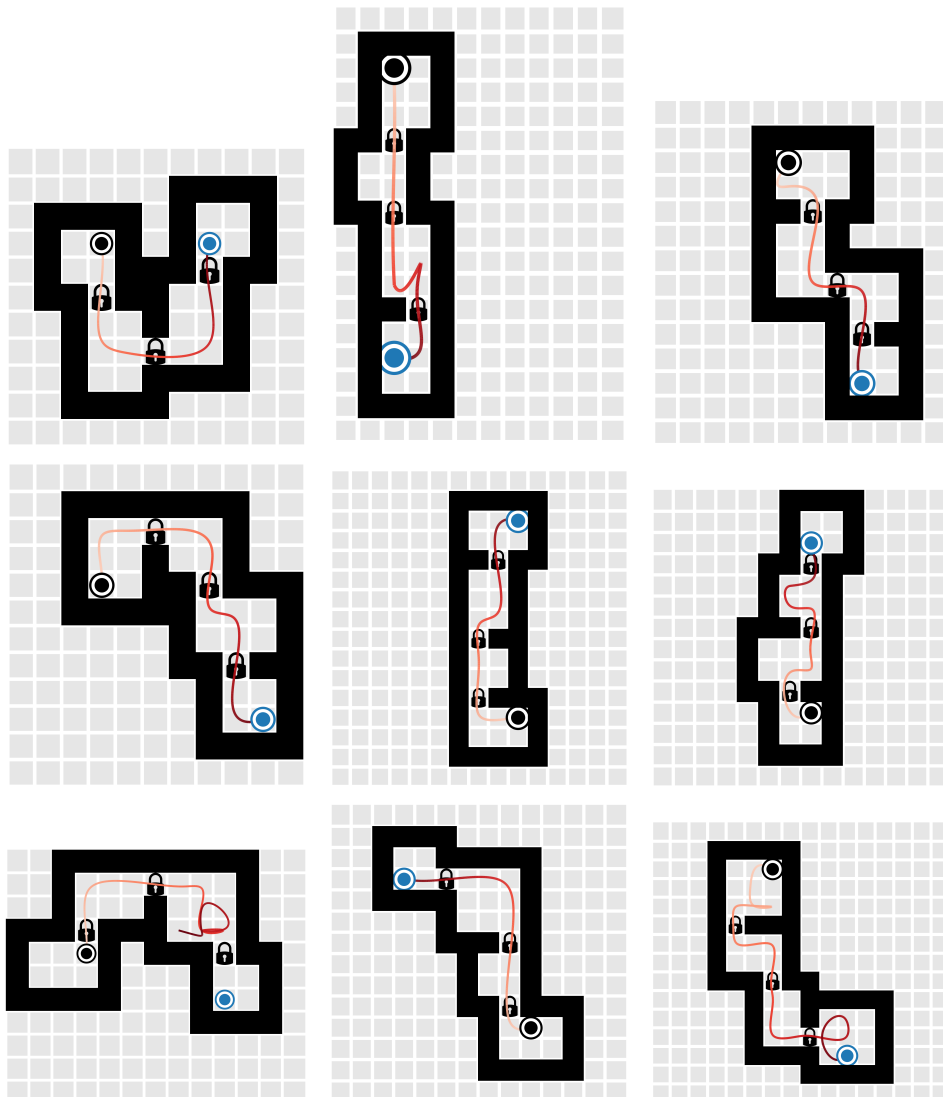
Figure B.1: **(Minigrid rollouts)** Example paths of the Trajectory Transformer planner in the `MiniGrid-MultiRoom-N4-S5`. Lock symbols indicate doors.

Appendix C

Diffuser Details

C.1 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.

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.

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 $\in [0.001, 0.01]$

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.

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).
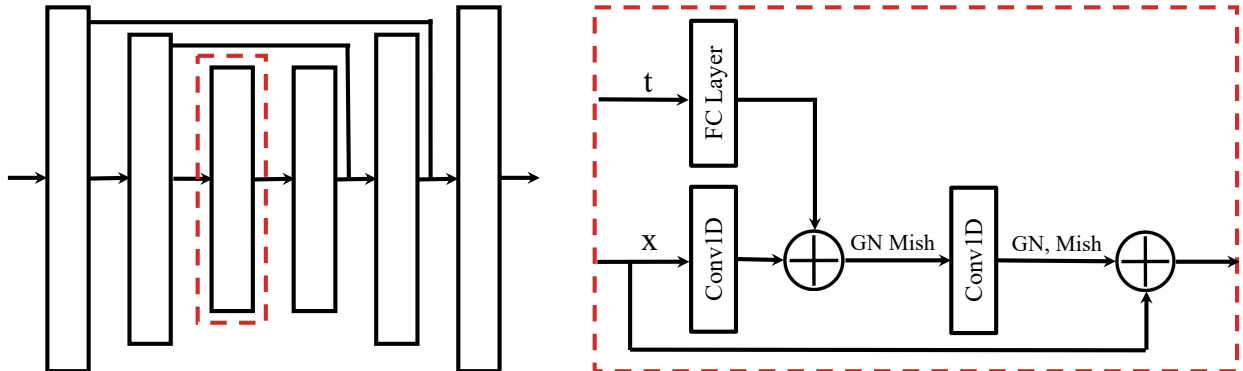
Figure C.1: **(Diffuser U-Net architecture)** Diffuser has a U-Net architecture with residual blocks consisting of temporal convolutions, group normalization, and Mish nonlinearities.

C.2 Test-time Flexibility

To guide Diffuser to stack blocks in specified configurations, we used two separate perturbation functions $h(\tau)$ 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 $h_{\text{match}}(\tau)$ as a per-timestep classifier determining whether a a state 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 $h_{\text{contact}}(\tau) = \sum_{i=0}^{64} -1 * \|\tau_{c_i} - 1\|^2$, where τ_{c_i} corresponds to the underlying dimension in state τ_{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.

C.3 Implementation Details

In this section we describe the architecture and record hyperparameters.

1. The architecture of Diffuser (Figure C.1) 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 $4e-05$ 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 32 in all locomotion tasks, 128 for block-stacking, 128 in `Maze2D / Multi2D U-Maze`, 265 in `Maze2D / Multi2D Medium`, and 384 in `Maze2D / Multi2D Large`.

5. We found that we could reduce the planning horizon for many tasks, but that the guide scale would need to be lowered (*e.g.*, to 0.001 for a horizon of 4 in the `halfcheetah` tasks) to accommodate. The configuration file in the open-source code demonstrates how to run with a modified scale and horizon.

6. We use $N = 20$ diffusion steps for locomotion tasks and $N = 100$ for block-stacking.

7. We use a guide scale of $\alpha = 0.1$ for all tasks except `hopper-medium-expert`, in which we use a smaller scale of 0.0001.

8. We used a discount factor of 0.997 for the return prediction $\mathcal{J}_\phi$, though found that above $\gamma = 0.99$ planning was fairly insensitive to changes in discount factor.

9. 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.