Receding-Horizon Control via Drifting Models

Daniele Fofano [*1] **Alessio Russo** [*2] **Alexandre Proutiere** [1]

Abstract

We study the problem of trajectory optimization in settings where the system dynamics are unknown and it is not possible to simulate trajectories through a surrogate model. When an offline dataset of trajectories is available, an agent could directly learn a trajectory generator by distribution matching. However, this approach only recovers the behavior distribution in the dataset, and does not in general produce a model that minimizes a desired cost criterion. In this work, we propose *Drifting MPC*, an offline trajectory optimization framework that combines drifting generative models with receding-horizon planning under unknown dynamics. The goal of Drifting MPC is to learn, from an offline dataset of trajectories, a conditional distribution over trajectories that is both supported by the data and biased toward optimal plans. We show that the resulting distribution learned by Drifting MPC is the unique solution of an objective that trades off optimality with closeness to the offline prior. Empirically, we show that Drifting MPC can generate near-optimal trajectories while retaining the one-step inference efficiency of drifting models and substantially reducing generation time relative to diffusion-based baselines.

1. Introduction

Trajectory optimization lies at the heart of control and robot autonomy. In many settings, however, computing an accurate control law through trajectory optimization is itself difficult because it requires a model of the environment that is sufficiently faithful for planning. While data-driven modeling (Willems et al., 2005; Tang and Daoutidis, 2022; Coulson et al., 2019; Russo and Proutiere, 2023) and system identification (Ljung, 1998) provide natural alternatives, in the presence of noise or nonlinearities, it may be difficult to construct a model of the system that is accurate enough to compute a control law. The system dynamics may be unknown or hard to identify precisely. In such cases, the agent must rely on an offline dataset of previously collected trajectories, often generated by heterogeneous and possibly suboptimal controllers, to learn an optimal action plan.

Reinforcement learning (Sutton and Barto, 2018) is a natural alternative in this setting. In particular, model-based RL (Polydoros and Nalpantidis, 2017; Argenson and Dulac-Arnold, 2021) seeks to learn a model of the system from data and then use that model for planning or policy improvement (Janner et al., 2019; Yu et al., 2020; Kidambi et al., 2020). This can be effective when the learned model is sufficiently accurate in the regions visited by the resulting controller. However, the quality of the control law then depends on the quality of the learned model over long horizons, and even small prediction errors may accumulate during rollouts and distort the optimization process (Foffano et al., 2025; Janner et al., 2019; Yu et al., 2020). This issue is particularly acute in offline settings, where the model must be learned from a fixed dataset and cannot be corrected through further interactions (Kidambi et al., 2020; Prudencio et al., 2023).

A different alternative is to bypass explicit model learning altogether and instead learn a conditional generator that proposes full trajectory plans directly from data (Janner et al., 2021; 2022). In this setting, one does not attempt to identify the underlying transition dynamics and then optimize through it. Rather, one learns a distribution over trajectories themselves, conditioned on the current state. This perspective is attractive for two reasons: (i) it amortizes computation so that planning at test time reduces to generating and scoring candidate trajectories; (ii) it shifts the learning problem from system identification to trajectory generation (Janner et al., 2021; Chen et al., 2021; Janner et al., 2022), which can be preferable in settings where accurate one-step prediction is difficult but the dataset still contains coherent long-horizon behavior. In principle, such

[1]Division of Decision and Control Systems, KTH Royal Institute of Technology, Stockholm, Sweden [2]Faculty of Computing and Data Sciences, Boston University, Boston, MA, USA. Correspondence to: Daniele Foffano <foffano@kth.se>, Alessio Russo <arusso2@bu.edu>.

Accepted at ICML 2026 Workshop on Decision-Making from Offline Datasets to Online Adaptation: Black-Box Optimization to Reinforcement Learning, Seoul, South Korea, 2026. Copyright 2026 by the author(s).

a model can be embedded inside a receding-horizon control strategy by repeatedly sampling candidate trajectories from the current state and executing the control action from the one with smallest cost.

Yet, this approach introduces a fundamental mismatch between modeling and optimization. A generator trained only by distribution matching will reproduce the behavior contained in the offline dataset, namely what the data-collecting controllers tended to do, but not necessarily what is optimal according to some cost criterion. The challenge is therefore to retain the computational advantages of direct trajectory generation while biasing the generated plans toward optimal solutions.

Diffusion-based planners address this challenge by combining trajectory generation with reward or cost guidance (Janner et al., 2022), and have shown that generative models can be effective tools for planning. Their main drawback is computational: they require multiple denoising steps at inference time, which makes them difficult to deploy in real-time control scenarios where the planner must be queried repeatedly. Drifting models provide a complementary alternative (Deng et al., 2026). They replace iterative denoising with a one-step pushforward generator trained through an attraction-repulsion field, making them attractive for fast receding-horizon planning. However, if applied directly to offline data, they still converge to the empirical behavior distribution, and therefore do not in general solve the trajectory optimization problem of interest.

In this work, we propose *Drifting MPC*, an offline trajectory optimization framework that combines drifting generative models with receding-horizon planning under unknown dynamics. The key idea is to modify the positive drift field so that the learned generator is trained toward an optimal target distribution rather than toward the raw dataset distribution. More precisely, we use an exponentially tilted offline prior, which increases the influence of optimal trajectories while preserving regularization toward the local support of the data. The learned generator is then used inside a best-of-M receding-horizon planner: given the current state and a cost query, it proposes several candidate trajectories, their known quadratic cost is evaluated, and the first control of the best candidate is executed.

We prove that our method is theoretically sound and we empirically validate it by comparing it with several baselines, showing that our method can produce optimal controls with a speed close to executing the closed form solution.

2. Related Work

Trajectory modeling for offline decision-making A closely related line of work views offline decision-making as a sequence or trajectory modeling problem. Decision Transformer casts offline reinforcement learning as return-conditioned sequence generation (Chen et al., 2021; Emmons et al., 2021), while Trajectory Transformer models trajectories autoregressively and performs planning in the learned sequence space (Janner et al., 2021). More broadly, conditional generative modeling has also been proposed as a general framework for offline decision-making (Ajay et al., 2023). These works share the same high-level motivation as ours: replace explicit dynamic programming or repeated model rollouts with a learned model over trajectories. Our setting, however, is different in two key respects. First, we target one-step trajectory generation rather than autoregressive generation. Second, instead of conditioning on desired return or using search over a sequence model, we bias the learned distribution by changing the positive distribution in the drifting objective itself.

Diffusion-based planning and control Diffusion models (Ho et al., 2020) have recently become a standard tool for generative planning. Diffuser formulates planning as conditional trajectory denoising and try to include constraints, or goal information through test-time guidance (Janner et al., 2022). Related diffusion-based methods have also been used as policy classes for offline reinforcement learning and robot control (Wang et al., 2023; Chi et al., 2024). These methods form the most natural baseline family for our work. The main difference is computational: diffusion planners rely on multiple denoising steps at inference time, whereas Drifting MPC aims to learn a one-step proposal model that can be queried repeatedly inside a receding-horizon loop.

Offline model-based planning and learning-augmented MPC Another nearby literature studies offline planning through learned dynamics models. Model-based offline reinforcement learning and planning methods such as MBOP, MOPO, MOReL, and MOPP learn a surrogate model from static data and then optimize through that model at test time (Argenson and Dulac-Arnold, 2021; Yu et al., 2020; Kidambi et al., 2020; Zhan et al., 2022). Closely related are learning-augmented MPC approaches, which use learned proposal distributions or sequence models to warm-start or regularize online planning (Sacks and Boots, 2022; Celestini et al., 2024; Russo and Proutiere, 2023; Tranos et al., 2023). Drifting MPC is related to this line of work in that it is also used inside a receding-horizon controller, but it differs in a fundamental way: it does not roll candidate trajectories through a learned model or solve a trajectory optimization problem online.

Drifting models and regularized control. Our method builds directly on Drifting Models, which replace iterative denoising of diffusion models with a single pushforward map (Deng et al., 2026). Standard drifting, however, matches the data distribution and therefore recovers the offline behavior prior rather than an objective-aware planning distribution. Drifting MPC addresses exactly this limitation, so that the generator is biased towards optimal trajectories. This idea is also closely related to the broader control-as-inference view of regularized optimal control, where optimal control distributions arise by exponentially tilting a prior with task-dependent rewards or costs (Levine, 2018).

3. Background and Problem Definition

In receding-horizon control at every decision step the controller must synthesize a sequence of control actions that minimize some cost criterion. When an accurate model is available, this problems fits naturally within a model-predictive control (MPC) framework. In the regime considered in this paper, however, we assume the agent does *not* have access to the underlying transition law. We now describe the model considered in the paper and the problem definition.

3.1. Setting

Model In the following we consider a discrete-time control system $\mathcal{M} = (\mathcal{X}, \mathcal{U}, f, \rho_0, H)$, where $\mathcal{X} \subseteq \mathbb{R}^{d_x}$ is the state space, $\mathcal{U} \subseteq \mathbb{R}^{d_u}$ is the control space, f is the unknown transition law over the next state, ρ_0 is the initial-state distribution, and $H \in \mathbb{N}$ is the planning horizon. Therefore, at each timestep the state evolves according to $x_{t+1} \sim f(\cdot \mid x_t, u_t)$, where u_t is the control action at timestep t, and the initial state is $x_0 \sim \rho$.

In the following, we write an H-step trajectory as

$$\tau = (x_0, u_0, x_1, u_1, \dots, x_{H-1}, u_{H-1}, x_H), \qquad x_0 = x,$$

where x is the initial state, and by $\tau_t = (x_0, u_0, \dots, x_t, u_t)$ its truncation to the first t transitions. We also denote by $\mathcal{T}_H(x)$ the set of all such H-step trajectories starting at x, and by $\mathcal{T}_H = \bigcup_{x \in \mathcal{X}} \mathcal{T}_H(x)$ the corresponding global trajectory space.

Trajectory cost In the following, we associate to each trajectory τ a cost. Specifically, the finite-horizon cost associated with a trajectory $\tau \in \mathcal{T}_H(x)$ is

$$J_x(\tau; \omega) := \sum_{t=0}^{H-1} \left(x_t^\top Q(\omega) x_t + u_t^\top R(\omega) u_t \right) + x_H^\top Q(\omega) x_H. \tag{1}$$

where $Q(\omega) \in \mathbb{R}^{d_x \times d_x}$ and $R(\omega) \in \mathbb{R}^{d_u \times d_u}$ are semi-definite positive matrices parametrized by a cost parameter $\omega = (q, r) \in \Omega$ in a compact set Ω. For simplicity, without loss of generality in the following we assume $Q(\omega) = \mathrm{diag}(q)$ and $R(\omega) = \mathrm{diag}(r)$ and

$$\Omega = [q_{\min}, q_{\max}]^{d_x} \times [r_{\min}, r_{\max}]^{d_u},$$

with non-negative bounds $q_{\min} \geq 0, q_{\max} \geq 0$.

3.2. Problem Definition

We are interested in deriving a data-driven receding-horizon control law to minimize the expected cost $J_x(\cdot; \omega)$ at any starting state x and cost parameter ω. Unlike previous work on data-driven methods, we consider a generative approach in which we learn a law $\mu_{x,\omega}$ over the set of trajectories $\mathcal{T}_H(x)$ for a given cost descriptor ω. In particular, our goal is not to learn just any law, but rather a law $\mu_{x,\omega}$ that minimizes the expected trajectory cost.

Objective In the following, for any fixed $(x, \omega) \in \mathcal{X} \times \Omega$ we consider the following optimization problem

$$\begin{aligned} &\min_{\mu \in \Delta(\mathcal{T}_H(x))} \mathbb{E}_{\tau \sim \mu} \left[J_x(\tau; \omega) \right] \\ &\text{s.t. } \mu\left(\mathrm{d}x_{t+1} \mid \tau_t\right) = f\left(\mathrm{d}x_{t+1} \mid x_t, u_t\right) \quad \text{a.s. } \forall t < H, \end{aligned} \tag{2}$$

where $\Delta(\mathcal{T}_H(x))$ denotes the set of probability measures over H-step trajectories starting at x.

Control law We propose to use a minimizer of Eq. (2) to implement a receding-horizon control law. Assuming the agent can compute a minimizer $\mu^\star_{x,\omega}$ at any (x, ω), then, at any timestep t after observing the state x, the agent can sample a trajectory $\tau \sim \mu^\star_{x,\omega}$ and execute the first control action u_0 in this trajectory. The agent then observes the next state and repeats the procedure.

Importantly, in this paper we rule out learning a model-based inner loop to approximate $\mu^\star$: therefore we cannot score candidate controls by simulating them forward. This motivates an algorithmic design that learns a full distribution over H-step trajectories (and not only actions). At the same time, we also need to find a distribution μ that minimizes the expected cost.

Offline trajectories Lastly, we assume the agent has access to an offline dataset of trajectories

$$\mathcal{D} = \{\tau_i\}_{i=1}^N, \tag{3}$$

where each τ_i is a horizon-H trajectory segment generated by some control law. In the following, we denote by $P_{\mathcal{D}} = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \delta_{\tau_i}$ the empirical distribution of trajectories in $\mathcal{D}$.

3.3. Drifting Models

Drifting models are one-step generative models that learn by iteratively transporting samples toward a desired target

distribution p (Deng et al., 2026). The goal of these models is to train a generator so that it produces a sample distributed according to p in a single forward pass. Henceforth, in contrast to diffusion models, which require multiple denoising steps at inference time, drifting models retain single-step generation, which makes them particularly appealing for receding-horizon control.

Mathematically, given some noise $\varepsilon \sim \mathcal{N}(0, I)$, a generator produces a sample $z = G_\theta(\varepsilon)$, which induces a conditional distribution q_θ. Then, at training time, the evolution of a sample z is governed by the following equation $z_{k+1} = z_k + V_{p,q_\theta}(z_k)$ where V_{p,q_θ} is a *drift field* that quantifies the shift. Following (Deng et al., 2026), we obtain that $V_{p,q_\theta} = 0$ when $p = q_\theta$ (it is also possible to give some sufficient conditions under which $V_{p,q_\theta} \approx 0$ implies $q_\theta \approx p$). Therefore, the objective is to train G_θ so that $\mathbb{E}[\|V_{p,q_\theta}\|] \approx 0$.

To define a drift field V_{p,q_θ}, the main idea is to ensure that it moves generated samples toward a chosen positive distribution p while repelling them from the current model distribution q_θ. To that aim, we define the drift field as

$$V_{p,q_\theta}(z) = V_p^+(z) - V_{q_\theta}^-(z). \tag{4}$$

where V_p^+ is the positive mean-shift field and $V_q^-(z)$ is the negative mean-shift field. To define these fields, we introduce a kernel that measures local similarity

$$k(z, z') = \exp\left(-\frac{\|z - z'\|_2^2}{T}\right),$$

where $T > 0$ is a temperature parameter. Then, the positive and negative mean-shift fields are defined as

$$V_p^+(z) = \frac{\mathbb{E}_{z^+\sim p}\left[k\left(z, z^+\right)\left(z^+ - z\right)\right]}{\mathbb{E}_{z^+\sim p}\left[k\left(z, z^+\right)\right]}, \tag{5}$$

$$V_{q_\theta}^-(z) = \frac{\mathbb{E}_{z^-\sim q_\theta}\left[k\left(z, z^-\right)\left(z^- - z\right)\right]}{\mathbb{E}_{z^-\sim q_\theta}\left[k\left(z, z^-\right)\right]}. \tag{6}$$

Training is performed with a fixed-point objective: if $z_\theta = G_\theta(\varepsilon)$, then the model is updated so that z moves toward its drifted target,

$$\mathcal{L}_{\text{drift}}(\theta) = \mathbb{E}_{z\sim q_\theta}\left[\|z - \text{sg}\left(z + V_{p,q_\theta}(z)\right)\|_2^2\right], \tag{7}$$

where $\text{sg}(\cdot)$ denotes stop-gradient. At equilibrium, the generated distribution matches the chosen positive distribution.

4. Method

Our method revolves around solving Eq. (2) using an offline dataset $\mathcal{D}$, and using the minimizer $\mu^\star$ to compute a receding-horizon control law. We propose Drifting MPC, a method based on Drifting Models (Deng et al., 2026) to learn a minimizer $\mu^\star$ for any (x, ω). The goal of Drifting MPC is to learn, from the offline dataset $\mathcal{D}$, a conditional generator that maps noise, the current state, and the cost parameter to a distribution over relative trajectories that is both supported by the data and skewed toward low-cost plans. The learned generator is then used as a proposal mechanism inside a best-of-M receding-horizon planner.

The central question is how to modify the drift modeling approach to learn $\mu^\star$ given an offline dataset $\mathcal{D}$. In fact, if one simply applied the drift modeling approach to learn a distribution $\mu \approx P_\mathcal{D}$, we do not necessarily have that μ minimizes Eq. (2). We propose to change the positive mean-shift field $V_{P_\mathcal{D}}^+$ in drift modeling so that the learned generator is not only faithful to the offline data, but also biased toward trajectories that are useful for control.

In the following subsection, we introduce a method to shift a (conditioned) positive drift field, so that we can approximately solve Eq. (2).

4.1. Conditionally Shifted Drift Fields

We now explain how to modify the drift modeling approach to learn shifted distributions conditioned on some query c. In the following we denote a planning query by $c = (x_0, \omega) \in \mathcal{X} \times \Omega$, and define a conditional generator as a parametrized model $G_\theta : \mathbb{R}^{d_\epsilon} \times \mathbb{R}^{2d_x+d_u} \to \mathbb{R}^{H\cdot(d_x+d_u)+d_x}$ that takes as input a conditioning query c and noise $\epsilon \sim p_\epsilon$ (with $p_\epsilon = \mathcal{N}(0, I)$). The genorator maps noise to samples $z = G_\theta(\epsilon, c)$: hence, the generator induces the conditional pushforward distribution

$$q_\theta(\cdot \mid c) = [G_\theta(\cdot, c)]_\# p_\epsilon.$$

4.1.1. Target Distribution

The goal of generative modeling is to train G_θ so that it induces some desired target distribution. Assume we have some prior distribution $p_0(\cdot \mid x_0)$ over trajectories with starting state x_0 and satisfying the transition law f. The key idea is to replace this prior with a cost-aware positive distribution. For an inverse temperature $\beta > 0$, define the tilted target

$$p_\beta(\mathrm{d}\tau \mid c) \propto \exp\left(-\beta J_{x_0}(\tau; \omega)\right) p_0(\mathrm{d}\tau \mid x_0). \tag{8}$$

where the weight $\exp\left(-\beta J_{x_0}(\tau; \omega)\right)$ down-weights trajectories with larger cost, and promotes trajectories with lower cost.

We can show than this distribution p_β solves a regularized version of Eq. (2).

Theorem 1 (Variational characterization of the tilted distribution)**.** *Fix $c = (x_0, \omega)$ and $\beta > 0$. Let $p_0(\cdot \mid x_0)$ be a*

reference distribution over trajectories and define

$$p_\beta(d\tau \mid c) = \frac{e^{-\beta J_{x_0}(\tau;\omega)}}{Z_\beta(c)} p_0(d\tau \mid x_0), \tag{9}$$

where $Z_\beta(c) = \int e^{-\beta J_{x_0}(\tau;\omega)} p_0(d\tau \mid x_0)$. *Then* $p_\beta(\cdot \mid c)$ *is the unique minimizer of*

$$\min_{p \ll p_0(\cdot|x_0)} \mathbb{E}_{\tau\sim p}\left[J_{x_0}(\tau;\omega)\right] + \frac{1}{\beta}\,\mathrm{KL}\left(p(\cdot \mid c) \| p_0(\cdot \mid x_0)\right). \tag{10}$$

Proof. For any $p \ll p_0(\cdot \mid x_0)$,

$$\log \frac{dp}{dp_\beta} = \log \frac{dp}{dp_0} + \beta J_{x_0}(\tau;\omega) + \log Z_\beta(c).$$

Taking expectation with respect to p yields

$$\mathrm{KL}(p\|p_\beta) = \mathrm{KL}(p\|p_0) + \beta \mathbb{E}_p[J_{x_0}(\tau;\omega)] + \log Z_\beta(c).$$

Rearranging gives

$$\mathbb{E}_p[J_{x_0}(\tau;\omega)] + \tfrac{1}{\beta}\big(\mathrm{KL}(p\|p_0) + \log Z_\beta(c)\big) = \tfrac{1}{\beta}\,\mathrm{KL}(p\|p_\beta).$$

The first term on the r.h.s. is constant in p, while the second is non-negative and vanishes only when $p = p_\beta$. □

Theorem 1 shows that p_β is not merely favoring low-cost trajectories: it is targeting the solution of a problem that trades off two competing objectives: minimizing control cost and remaining close to the offline trajectory prior. The distribution is characterized by β, which defines how aggressively the the tilted distribution shifts away from the prior p_0 towards lost-cost trajectories. From a practical perspective, early in training small values of β may be beneficial, as it recovers a behavior-like prior and stabilize optimization. As training proceeds, larger values of β gradually transform the same local prior into a sharper, more optimization-oriented target.

4.1.2. Tilting Lemma

In the following result, we show how p_β can be learned using drift modeling. The idea is to tilt the positive drift field $V^+_{p_0}$. Recall that to define a drifting field we require a kernel that measures local similarity. We use a Gaussian kernel

$$k(\tau,\tau') = \exp\left(-\frac{\|\tau-\tau'\|_2^2}{T}\right),$$

where $T > 0$ is a temperature parameter. Then, given an initial state x_0 and a trajectory $\tau \in \mathcal{T}_H(x_0)$, the positive mean-shift field for p_0 in x_0 is defined as

$$V^+_{p_0}(\tau;x_0) = \frac{\mathbb{E}_{\tau^+\sim p_0(\cdot|x_0)}\left[k\left(\tau,\tau^+\right)\left(\tau^+-\tau\right)\right]}{\mathbb{E}_{\tau^+\sim p_0(\cdot|x_0)}\left[k\left(\tau,\tau^+\right)\right]}, \tag{11}$$

and similarly for p_β we have

$$V^+_{p_\beta}(\tau;x_0) = \frac{\mathbb{E}_{\tau^+\sim p_\beta(\cdot|x_0)}\left[k\left(\tau,\tau^+\right)\left(\tau^+-\tau\right)\right]}{\mathbb{E}_{\tau^+\sim p_\beta(\cdot|x_0)}\left[k\left(\tau,\tau^+\right)\right]}. \tag{12}$$

Using the simple fact that $p_\beta \propto \exp(-\beta J_{x_0})p_0$, we find the following immediate result implying that it is sufficient to tilt the original mean-shift drift.

Lemma 1 (Tilting)**.** *Fix* $c = (x_0,\omega)$ *and let* $p_0(\cdot|x_0)$ *be a reference distribution over trajectories. Define the weight* $w_\beta(\tau;c) = \exp(-\beta J_{x_0}(\tau;\omega))$. *Define the weighted mean-shifted operator:*

$$V^+_{p_0}(\tau;\beta,c) \coloneqq \frac{\mathbb{E}_{\tau'\sim p_0(\cdot|x_0)}\left[w_\beta(\tau';c)k(\tau,\tau')(\tau'-\tau)\right]}{\mathbb{E}_{\tau'\sim p_0(\cdot|x_0)}\left[w_\beta(\tau';c)k(\tau,\tau')\right]}.$$

We have that

$$V^+_{p_\beta}(\tau;x_0) = V^+_{p_0}(\tau;\beta,c) \quad p_0 - a.e.$$

Proof. By definition of p_β, for any integrable h we have

$$\begin{aligned}\mathbb{E}_{\tau'\sim p_\beta}[h(\tau)] &= \int h(\tau')\frac{w_\beta(\tau';c)}{Z}p_0(\mathrm{d}\tau' \mid x_0),\\ &= \frac{1}{Z}\mathbb{E}_{\tau'\sim p_0(\cdot|x_0)}[w(\tau';c)h(\tau')].\end{aligned}$$

Apply this identity in $V^+_{p_\beta}(\tau;x_0)$ with $h(\tau') = k(\tau,\tau')(\tau'-\tau)$ for the numerator and $h(\tau') = k(\tau,\tau')$ for the denominator. The factor $1/Z$ cancels between numerator and denominator, yielding the claim. □

This result is what makes Drifting MPC implementable. It shows that the algorithm only needs relative importance weights of the form $e^{-\beta J}$, not samples from a globally normalized target distribution. The proposition therefore provides the formal bridge between the ideal cost-aware target and the practical minibatch-level computation carried out during training.

4.1.3. Drift Loss

We are now ready to define the drift loss used to train the generator G_θ. First, we define $\hat{p}_0(\cdot \mid x_0)$ as an empirical prior computed using the offline data $\mathcal{D}$.

We use the following empirical prior

$$\hat{p}_0(\cdot \mid x_0) \coloneqq \sum_{i=1}^{N} \alpha_i(x_0)\,\delta_{\tau_i}(\cdot), \tag{13}$$

where α_i is defined as the following normalized weight

$$\alpha_i(x_0) \propto k_x(x_0,x_i)\,\mathbf{1}\{i \in \mathcal{N}_K(x_0)\},$$

where x_i is the initial state of trajectory τ_i. The kernel $k_x : \mathbb{R}^{d_x} \times \mathbb{R}^{d_x} \to [0,\infty)$ measures the similarity between

x_0 and x_i and $\mathcal{N}_K(x_0)$ is the set of K nearest neighbors retrieved from the offline dataset $\mathcal{D}$ defined according to some distance. Intuitively, (13) focuses the prior generator on trajectories that are compatible with x_0, and with $K_x(x_0, x_i) = \mathbf{1}_{\{x_0 = x_i\}}$ we retrieve the true empirical prior.

Then, for a given $c = (x_0, \omega)$, we define the empirical drift loss by the following fixed-point objective

$$\mathcal{L}_{\text{drift}}(\theta; c) = \mathbb{E}\left[\left\|\tau - \text{sg}\left(\tau + \widehat{V}_{\hat{p}_0, q_\theta}(\tau; \beta, c)\right)\right\|_2^2\right], \tag{14}$$

where the empirical drift field $\widehat{V}_{\hat{p}_0, q_\theta}(\tau; \beta, c)$ for $\tau \sim q_\theta(\cdot \mid c)$ is defined as

$$\widehat{V}_{\hat{p}_0, q_\theta}(\tau; \beta, c) = \widehat{V}^+_{\hat{p}_0}(\tau; \beta, c) - \widehat{V}^-_{q_\theta}(\tau; c),$$

with positive drift field $\widehat{V}^+_{\hat{p}_0}(\tau; \beta, c)$ and negative drift field $\widehat{V}^-_{q_\theta}(\tau; c)$. These fields are defined as follows

$$\widehat{V}^+_{\hat{p}_0}(\tau; \beta, c) \coloneqq \mathbb{E}_{\mathcal{B}^+, \tau^+}\left[\tilde{k}^+_{\mathcal{B}, \beta}(\tau, \tau^+; c)(\tau^+ - \tau)\right],$$
$$\widehat{V}^-_{q_\theta}(\tau; c) \coloneqq \mathbb{E}_{\mathcal{B}^-, \tau^-}\left[\tilde{k}^-_{\mathcal{B}}(\tau, \tau^-; c)(\tau^- - \tau)\right]$$

with $\tau^+ \sim \hat{p}_0(\cdot \mid x_0)$, $\tau^- \sim q_\theta(\cdot \mid c)$. The batch $\mathcal{B} = \{\mathcal{B}^+, \mathcal{B}^-\}$ is used to empirically approximate the weights, and it contains positive samples

$$\mathcal{B}^+ = (\tau_i^+)_{i=1}^K,\ \tau_i^+ \sim \hat{p}_0(\cdot \mid x_0),$$

and negative samples

$$\mathcal{B}^- = (\tau_i^-)_{i=1}^M,\ \tau_i^- \sim q_\theta(\cdot \mid c).$$

Then, the normalized positive weights $\tilde{k}^+_{\mathcal{B}}(\tau, \tau^+)$ for the positive field is defined as

$$\tilde{k}^+_{\mathcal{B}, \beta}(\tau, \tau^+; c) = \frac{e^{-\beta \widetilde{J}_i(\omega)}\, k(\tau, \tau^+)}{\sum_{\tau_j \in \mathcal{B}^+} e^{-\beta \widetilde{J}_j(\omega)}\, k(\tau, \tau_j)}. \tag{15}$$

where we used Lemma 1 and $\widetilde{J}_i(\omega)$ denotes the relabeled cost of the i-th retrieved trajectory in $\mathcal{B}^+$ under the query parameter ω. Lastly, the normalized weight for the negative field is

$$\tilde{k}^-_{\mathcal{B}}(\tau, \tau^-; c) = \frac{k(\tau, \tau^-)}{\sum_{\tau_j \in \mathcal{B}^- \setminus \{\tau\}} k(\tau, \tau_j)}. \tag{16}$$

Hence, Equation (14) moves each generated sample toward a local, cost-aware mean shift of the offline data while repelling it from the current model distribution.

4.2. Full algorithm

We now describe how Drifting MPC is trained in practice and how it is used at test time.

Algorithm 1 Training Drifting MPC

Require: offline dataset $\mathcal{D} = \{\tau_i\}_{i=1}^N$; generator G_θ; neighborhood size K; negative batch size M; query distribution $\hat{\rho}_0 \times \text{Unif}(\Omega)$; inverse-temperature schedule β

for each stochastic gradient step **do**
  Sample a batch of queries $c_b = (x_0^{(b)}, \omega^{(b)})$, $b = 1, \ldots, B$, with $x_0^{(b)} \sim \hat{\rho}_0$ and $\omega^{(b)} \sim \text{Unif}(\Omega)$.
  for $b = 1, \ldots, B$ **do**
    Construct positive batch $\mathcal{B}_b^+ = (\tau_i^+)_{i=1}^K$ from $\hat{p}_0(\cdot \mid x_0^{(b)})$ and compute $\widetilde{J}_i(\omega^{(b)})$ for each $\tau_i^+ \in \mathcal{B}_b^+$.
    Sample $\epsilon_1, \ldots, \epsilon_M \sim \mathcal{N}(0, I)$ and construct a negative batch $\mathcal{B}_b^- = (\tau_1^-, \ldots, \tau_M^-)$, $\tau_j^- = G_\theta(\epsilon_j, c_b)$.
    for $j = 1, \ldots, M$ **do**
      Compute the fields $\widehat{V}^+_{\hat{p}_0}(\tau_j^-; \beta, c_b), \widehat{V}^-_{q_\theta}(\tau_j^-; c_b)$ using Eqs. (15) and (16) and set
      $$\widehat{V}_{\hat{p}_0, q_\theta}(\tau_j^-; \beta, c_b) = \widehat{V}^+_{\hat{p}_0}(\tau_j^-; \beta, c_b) - \widehat{V}^-_{q_\theta}(\tau_j^-; c_b)$$
    end for
    Form the empirical query loss
    $$\widehat{\mathcal{L}}_{\text{drift}}(\theta; c_b) = \frac{1}{M}\sum_{j=1}^M \|\delta_j\|_2^2,$$
    $$\delta_j \coloneqq \tau_j^- - \text{sg}\Big(\tau_j^- + \widehat{V}_{\hat{p}_0, q_\theta}(\tau_j^-; \beta, c_b)\Big).$$
  end for
  update θ by a gradient step on $\frac{1}{B}\sum_{b=1}^B \widehat{\mathcal{L}}_{\text{drift}}(\theta; c_b)$
end for

Training phase The training objective is obtained by averaging the conditional drift loss Eq. (14) over a distribution of planning queries c. To that end, let

$$\hat{\rho}_0 \coloneqq \frac{1}{N}\sum_{i=1}^N \delta_{x_i}$$

denote the empirical distribution of initial states in the offline dataset, where x_i is the initial state of trajectory τ_i. At each training step, we sample an initial state $x_0 \sim \hat{\rho}_0$ and independently sample a cost parameter $\omega \sim \text{Unif}(\Omega)$. This induces a random query $c = (x_0, \omega)$. Sampling queries in this way amounts to a form of *meta-training*: rather than learning a generator for a single fixed objective, the model is trained over a family of problems indexed by both the current state and the cost parameter. The generator therefore learns an amortized map from queries to low-cost trajectory proposals.

Formally, the training objective is

$$\mathcal{L}_{\text{train}}(\theta) = \mathbb{E}_{x_0 \sim \hat{\rho}_0,\, \omega \sim \text{Unif}(\Omega)}\left[\mathcal{L}_{\text{drift}}(\theta; (x_0, \omega))\right], \tag{17}$$

Algorithm 2 Receding-horizon planning with Drifting MPC

Require: current state x; cost parameter ω; trained generator G_θ; number of candidates M_{plan}

1: Construct the query $c = (x, \omega)$
2: Sample $m = 1, \dots, M_{\text{plan}}$ candidate trajectories from the generator $\tau^{(m)} = G_\theta(\epsilon_m, c),\ \epsilon_m \sim \mathcal{N}(0, I)$, and evaluate their cost

$$C_m = J_x(\tau^{(m)}; \omega).$$

3: Choose $m^\star = \arg\min_{1 \le m \le M_{\text{plan}}} C_m$
4: Execute the first control of $\tau^{(m^\star)}$, observe the next state, and repeat.

where $\mathcal{L}_{\text{drift}}(\theta; c)$ is the conditional fixed-point loss in Eq. (14). In practice, Eq. (17) is optimized by stochastic gradient descent using Monte Carlo approximations of both the positive and negative drift fields.

For a sampled query $c = (x_0, \omega)$, we first construct a local positive batch

$$\mathcal{B}^+ = (\tau_1^+, \dots, \tau_K^+), \qquad \tau_i^+ \sim \hat{p}_0(\cdot \mid x_0),$$

by retrieving trajectories whose initial states are close to x_0 according to the weights $\alpha_i(x_0)$ in Eq. (13). These trajectories are then *relabeled* using the sampled cost parameter ω, producing relabeled costs

$$\widetilde{J}_i(\omega) = J_{x_i}(\tau_i^+; \omega),$$

which enter the positive weights in Eq. (15). Next, we sample a negative batch

$$\mathcal{B}^- = (\tau_1^-, \dots, \tau_M^-), \qquad \tau_j^- = G_\theta(\epsilon_j, c),\ \epsilon_j \sim \mathcal{N}(0, I),$$

from the current generator. The empirical positive and negative drift fields are then computed from $\mathcal{B}^+$ and $\mathcal{B}^-$ using Eqs. (15) and (16), and the loss Eq. (14) is evaluated on the generated samples. The full training procedure is summarized in Algorithm 1.

Inference and receding-horizon control Once training is complete, the generator is used as a one-step proposal mechanism inside a best-of-M MPC loop (the corresponding test-time procedure is summarized in Algorithm 2). Given the current state x and a query cost parameter ω, we form the planning query $c = (x, \omega)$ and sample

$$\tau^{(m)} = G_\theta(\epsilon_m, c), \quad \epsilon_m \sim \mathcal{N}(0, I), \quad m = 1, \dots, M_{\text{plan}}.$$

Each sampled trajectory is then evaluated under the true objective using $C_m = J_x(\tau^{(m)}; \omega)$. The planner selects the lowest-cost candidate (tie breaks arbitrarily),

$$m^\star = \arg\min_{1 \le m \le M_{\text{plan}}} C_m,$$

and executes only the first control of $\tau^{(m^\star)}$. After the next state is observed, the whole procedure is repeated. This yields a receding-horizon controller that combines an amortized, cost-aware trajectory generator with online selection among a small number of sampled plans.

For this type of inference mechanism, we are able to provide the following Best-of-M guarantee over T steps. We consider a fix ω cost, and introduce the following set of δ-optimal trajectories for $\delta > 0$:

$$\mathcal{A}_\delta(x, \omega) = \{\tau \in \mathcal{T}_H(x) : J_x(\tau; \omega) \le J_x^\star(\omega) + \delta\}$$

for $c = (x, \omega)$ and $J_x^\star(\omega) := \operatorname{ess\,inf}_{\tau \in \mathcal{T}_H(x)} J_x(\tau; \omega)$, where the essential infimum is taken with respect to the underlying trajectory law induced by (ρ_0, f). We also let $\hat{\tau}_t \in \arg\min_{m=1,\dots,M_{\text{plan}}} J_{x_t}(\tau_t^{(m)}; \omega)$ be the trajectory selected by the planner, with $(\tau_t^{(i)})_i$ sampled i.i.d. from $q_\theta(\cdot \mid x_t, \omega)$, and let

$$E_{\delta,t} = \{J_{x_t}(\hat{\tau}_t; \omega) > J_{x_t}^\star(\omega) + \delta\}$$

be the event that the planner is δ-suboptimal at timestep t.

Then, the following theorem gives a best-of-M guarantee for the receding-horizon planner induced by Drifting MPC.

Theorem 2. *Fix $\omega \in \Omega$, and let p_β be as in Eq.* (9)*. Assume there exists $\epsilon, \eta > 0$ such that*

$$\operatorname{ess\,sup}_{x \in \mathcal{X}} d_{\text{TV}}(q_\theta(\cdot \mid x, \omega), p_\beta(\cdot \mid x, \omega)) \le \varepsilon(\omega),$$
$$\operatorname{ess\,inf}_{x \in \mathcal{X}} p_\beta(A_\delta(x, \omega) \mid x, \omega) \ge \eta(\omega) > 0.$$

Then, for any $t \in \{0, \dots, T-1\}$ we have

$$\mathbb{P}\left(\bigcup_{t=0}^{T-1} E_{\delta,t} \,\middle|\, \omega\right) \le T \exp\left(-M_{\text{plan}} \max(0, \eta(\omega) - \varepsilon(\omega))\right). \tag{18}$$

Proof. Let $\{\mathcal{F}_t\}_{t \ge 1}$ be the filtration of the history up to step t, and let $c_t = (x_t, \omega)$ be the query at step t. Fix a timestep $t \ge 0$. Given the current query c_t, the candidates $(\tau_t^{(i)})_{i=1}^{M_{\text{plan}}}$ are sampled i.i.d. from $q_\theta(\cdot \mid c_t)$. Hence,

$$\begin{aligned}\mathbb{P}(E_{\delta,t} \mid \mathcal{F}_t, \omega) &= \prod_{m=1}^{M_{\text{plan}}} \mathbb{P}\left(\tau_t^{(m)} \notin A_\delta(x_t, \omega) \mid \mathcal{F}_t, \omega\right), \\ &= \left[1 - q_\theta(A_\delta(x_t, \omega) \mid c_t)\right]^{M_{\text{plan}}}.\end{aligned}$$

Now, since $d_{\text{TV}}(q_\theta(\cdot \mid c_t), p_\beta(\cdot \mid c_t)) = \sup_{\text{meas. } A} |q_\theta(A \mid c_t) - p_\beta(A \mid c_t)|$, it follows that for any measurable event A we have

$$q_\theta(A \mid c_t) \ge [p_\beta(A \mid c_t) - d_{\text{TV}}(q_\theta(\cdot \mid c_t), p_\beta(\cdot \mid c_t))]_+,$$

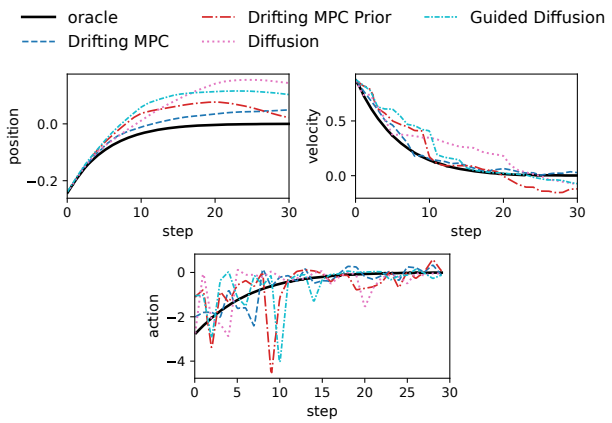

Figure 1. Rollouts obtained for the different models ($H = 30$).

where $[x]_+ = \max(x, 0)$. Applying this result with $A = A_\delta(x_t, \omega)$ we get

$$\begin{aligned}
&\mathbb{P}(E_{\delta,t} \mid \mathcal{F}_t, \omega) \\
&\quad \le [1 - p_\beta(A \mid c_t) + d_{\mathrm{TV}}(q_\theta(\cdot \mid c_t), p_\beta(\cdot \mid c_t))]_+^{M_{\mathrm{plan}}}, \\
&\quad \le \exp\left\{-M_{\mathrm{plan}}\left[p_\beta(A \mid c_t) - d_{\mathrm{TV}}(q_\theta(\cdot \mid c_t), p_\beta(\cdot \mid c_t))\right]_+\right\} \\
&\quad \le \exp\left\{-M_{\mathrm{plan}}\left[\eta(\omega) - \varepsilon(\omega)\right]_+\right\}.
\end{aligned}$$

The conclusion follows from a tower rule argument and a union bound over timesteps. □

In this result the quantity η measures how much probability mass the tilted target p_β assigns to δ-optimal trajectories, while ε measures how closely the learned generator q_θ matches p_β in total variation. Whenever $\eta > \varepsilon$, the probability of making a δ-suboptimal planning decision decays exponentially in the number of sampled candidates M_{plan}. This result shows that larger planning budgets and better approximation of p_β directly improve the reliability of the closed-loop planner, ensuring its δ-optimality.

5. Numerical Results

In this section, we present numerical experiments illustrating the advantages of our approach on a dynamical system.

Environment, dataset, and oracle The benchmark environment is the one-dimensional mass-spring-damper system

$$\dot{p} = v, \qquad \dot{v} = -\frac{k_s}{m}p - \frac{c}{m}v + \frac{1}{m}u,$$

which is discretized exactly under zero-order hold before being used for dataset collection and oracle evaluation. The default physical parameters in code are $m = 1.0$, $k_s = 1.0$, $c = 0.2$, and $\Delta t = 0.05$. The planning horizon is set equal to the episode length, and we consider values of 30, 50, and 100. Initial states are sampled uniformly from the box $[-2, 2] \times [-2, 2]$. Offline trajectories are collected by a mixture of controllers: a finite-horizon LQR oracle with optional action noise (10% of the dataset), a noisy PD controller (10%), and a smooth random open-loop controller (80%). The oracle benchmark is a finite-horizon LQR computed by backward Riccati recursion using the true discretized linear dynamics.

Table 1. Cost (mean ± SE and median [IQR]) and rollout time (mean ± SE) via BCa bootstrap (10 000 resamples).

Method	Avg Cost	Median Cost [IQR]	Avg Time [ms]
Horizon 30			
Oracle	90.1 ± 12.9	$46.5\ [16, 114]$	13.3 ± 0.1
Drift MPC	105.8 ± 14.2	$54.3\ [21, 127]$	29.8 ± 0.9
Drift Prior	116.9 ± 15.6	$60.1\ [22, 158]$	28.5 ± 0.2
Diffusion	119.6 ± 16.2	$58.2\ [21, 143]$	1623 ± 3.3
Guided Diffusion	106.4 ± 14.6	$51.4\ [18, 133]$	2068 ± 3.7
Horizon 50			
Oracle	107.3 ± 13.6	$49.5\ [21, 126]$	36.4 ± 0.2
Drift MPC	131.5 ± 15.6	$69.0\ [30, 154]$	54.3 ± 0.7
Drift Prior	169.2 ± 21.4	$83.2\ [39, 193]$	52.4 ± 0.2
Diffusion	201.9 ± 22.7	$122.8\ [54, 246]$	2707 ± 4.6
Guided Diffusion	623.5 ± 201.5	$67.8\ [34, 161]$	3467 ± 5.9
Horizon 100			
Oracle	93.2 ± 9.5	$65.3\ [24, 122]$	141 ± 0.4
Drift MPC	122.6 ± 11.3	$85.2\ [40, 161]$	135 ± 0.8
Drift Prior	168.1 ± 16.1	$116.4\ [60, 206]$	134 ± 0.5
Diffusion	8647 ± 694	$7080\ [3194, 12035]$	5514 ± 8.5
Guided Diffusion	27317 ± 6192	$1565\ [441, 13580]$	7055 ± 10.8

Implemented baselines We compare the proposed method, Drifting MPC, with 3 baselines:

- `Drifting Prior`: a drifting generator conditioned only on x_0.
- `Diffusion`: a DDPM-style (Ho et al., 2020) trajectory generator, without cost conditioning at test time.
- `Guided Diffusion`: the same model as in the Diffusion (Janner et al., 2022) baseline, but now equipped with classifier guidance at test time. The guidance signal is the cumulative trajectory cost, as in the Diffuser architecture.[1]

Every method has been trained on the same dataset, using 500 training epochs. Diffusion methods use 64 denoising steps.

Results Our numerical results show that Drifting MPC consistently achieves the best overall performance among

[1] Since we assume that the cost function is available at test time, the gradient of the cumulative cost can be computed in closed form, eliminating the need to train an additional classifier.

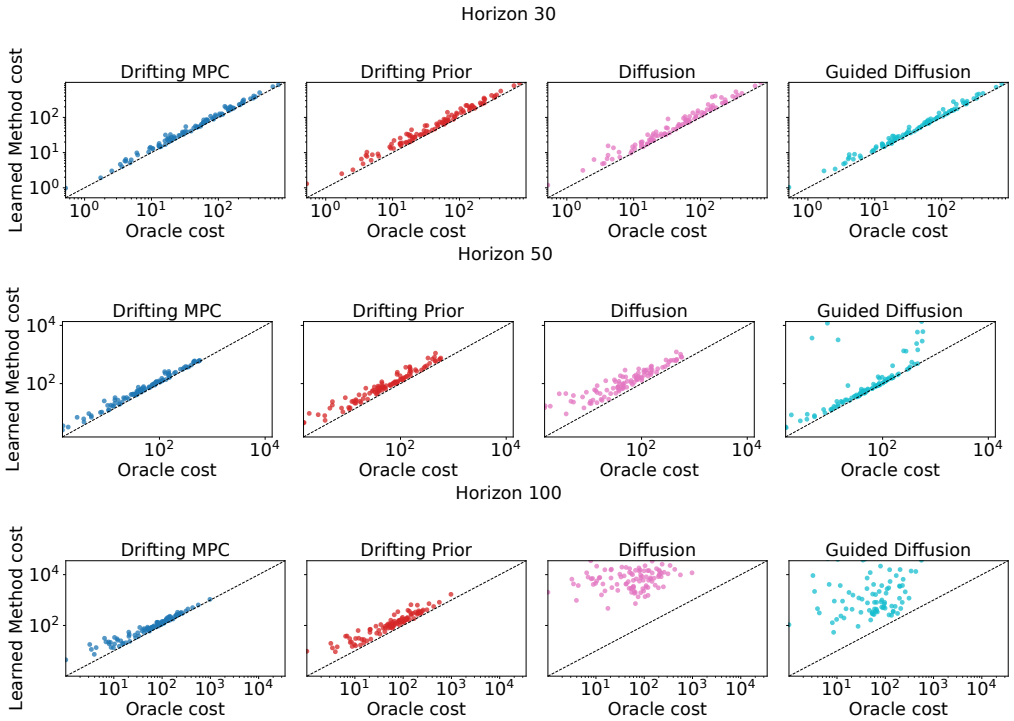

Figure 2. Scatter plots comparing the cost of 100 rollouts against the Oracle for horizons $H \in \{30, 50, 100\}$.

the learned methods. In Fig. 1, its rollout closely matches the oracle, unlike Drifting Prior and the diffusion baselines. Table 1 confirms that Drifting MPC attains substantially lower cost while remaining much faster than diffusion-based planners, and the scatter plots in Fig. 2 show that its performance is not only better on average but also more consistent, with costs concentrated near the oracle across episodes. While Oracle and Drifting methods maintain similar performance across horizons, both Diffusion baselines degrade significantly as the horizon grows. Their median costs are notably lower, however, indicating that a few catastrophic rollouts skew the mean; we attribute this to insufficient training epochs (and likely too few denoising steps) for convergence. This highlights that Drifting converges to a near-optimal solution faster than the diffusion baselines. Finally, Drifting methods generate rollouts within the same order of magnitude as the oracle, whereas Diffusion methods can be generally much slower due to repeated denoising steps and, for guided diffusion, the per-step classifier-guidance gradient computation.

6. Conclusions

We introduced Drifting MPC, an offline trajectory optimization framework that combines one-step drifting generative models with receding-horizon planning by tilting the learned trajectory distribution toward low-cost trajectories while remaining supported by offline data. This establishes a principled connection between trajectory generation and regularized optimal control, and yields a planner that can be queried efficiently at test time. Our experiments show that Drifting MPC outperforms both the drifting prior and diffusion-based baselines, achieving near-oracle performance while preserving the computational advantage of one-step generation.Future work should focus on a broader investigation of alternative guidance mechanisms.

References

A. Ajay, Y. Du, A. Gupta, J. Tenenbaum, T. Jaakkola, and P. Agrawal. Is Conditional Generative Modeling all you need for Decision-Making?, July 2023. arXiv:2211.15657 [cs].

A. Argenson and G. Dulac-Arnold. Model-Based Offline Planning, Mar. 2021. arXiv:2008.05556 [cs].

D. Celestini, D. Gammelli, T. Guffanti, S. D'Amico, E. Capello, and M. Pavone. Transformer-based model predictive control: Trajectory optimization via sequence modeling. *IEEE Robotics and Automation Letters*, 9(11): 9820–9827, 2024.

L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in neural information processing systems*, volume 34, pages 15084–15097, 2021.

C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion, Mar. 2024. arXiv:2303.04137 [cs].

J. Coulson, J. Lygeros, and F. Dörfler. Data-enabled predictive control: In the shallows of the DeePC. In *2019 18th European control conference (ECC)*, pages 307–312. IEEE, 2019.

M. Deng, H. Li, T. Li, Y. Du, and K. He. Generative Modeling via Drifting, Feb. 2026. arXiv:2602.04770 [cs].

S. Emmons, B. Eysenbach, I. Kostrikov, and S. Levine. Rvs: What is essential for offline rl via supervised learning? *arXiv preprint arXiv:2112.10751*, 2021.

D. Foffano, A. Russo, and A. Proutiere. Adversarial Diffusion for Robust Reinforcement Learning, Dec. 2025. arXiv:2509.23846 [cs].

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

M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. In *Advances in neural information processing systems*, volume 32, 2019.

M. Janner, Q. Li, and S. Levine. Offline reinforcement learning as one big sequence modeling problem. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in neural information processing systems*, volume 34, pages 1273–1286. Curran Associates, Inc., 2021.

M. Janner, Y. Du, J. Tenenbaum, and S. Levine. Planning with diffusion for flexible behavior synthesis. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, *Proceedings of the 39th international conference on machine learning*, volume 162 of *Proceedings of machine learning research*, pages 9902–9915. PMLR, July 2022.

R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims. Morel: Model-based offline reinforcement learning. In *Advances in neural information processing systems*, volume 33, pages 21810–21823, 2020.

S. Levine. Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review, May 2018. arXiv:1805.00909 [cs].

L. Ljung. System identification. In *Signal analysis and prediction*, pages 163–173. Springer, 1998.

A. S. Polydoros and L. Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

R. F. Prudencio, M. R. Maximo, and E. L. Colombini. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *IEEE transactions on neural networks and learning systems*, 35(8):10237–10257, 2023.

A. Russo and A. Proutiere. Tube-based zonotopic data-driven predictive control. In *2023 american control conference (ACC)*, pages 3845–3851, 2023. doi: 10.23919/ACC55779.2023.10156056.

J. Sacks and B. Boots. Learning Sampling Distributions for Model Predictive Control, Dec. 2022. arXiv:2212.02587 [cs].

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

W. Tang and P. Daoutidis. Data-driven control: Overview and perspectives. In *2022 American control conference (ACC)*, pages 1048–1064. IEEE, 2022.

D. Tranos, A. Russo, and A. Proutiere. Self-tuning tube-based model predictive control. In *2023 american control conference (ACC)*, pages 3626–3632, 2023. doi: 10.23919/ACC55779.2023.10155796.

Z. Wang, J. J. Hunt, and M. Zhou. Diffusion Policies as an Expressive Policy Class for Offline Reinforcement Learning, Aug. 2023. arXiv:2208.06193 [cs].

J. C. Willems, P. Rapisarda, I. Markovsky, and B. L. De Moor. A note on persistency of excitation. *Systems & Control Letters*, 54(4):325–329, 2005.

T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization. In *Advances in neural information processing systems*, volume 33, pages 14129–14142, 2020.

X. Zhan, X. Zhu, and H. Xu. Model-Based Offline Planning with Trajectory Pruning, Apr. 2022. arXiv:2105.07351 [cs].