INFERENCE TIME POLICY OPTIMIZATION FOR OFFLINE RL WITH DIFFERENTIABLE WORLD MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Offline Reinforcement Learning (RL) learns optimal policies from fixed datasets, training a policy once and deploying it at inference time without further refinement. Inspired by model predictive control (MPC), we introduce an inference time adaptation framework that utilizes a pretrained policy along with a learned world model. While existing world model and diffusion-planning methods use learned dynamics to generate imagined trajectories during training, or to sample candidate plans at inference time, they do not use inference-time information to *optimize* the policy parameters on the fly. In contrast, our design is a Differentiable World Model (DWM) pipeline that enables end-to-end gradient computation through imagined rollouts for *inference time policy optimization* (ITPO). We evaluate our algorithm on D4RL continuous-control benchmarks (MuJoCo locomotion tasks and AntMaze), and show that exploiting inference-time information to optimize the policy parameters yields consistent gains over strong offline RL baselines. Inference-time adaptation, however, is expensive: rollout generation and backpropagation dominate per-step compute. We study this tradeoff explicitly, showing that a suitable tilted version of one-step MeanFlow sampler recovers much of the gains at a fraction of the cost.

1 INTRODUCTION

Offline reinforcement learning learns a policy in a Markov decision process from a fixed dataset of previously collected trajectories, without any additional interaction with the environment (Levine et al., 2020; Figueiredo Prudencio et al., 2024; Ernst et al., 2005; Fujimoto et al., 2019; Liu et al., 2020). This setting arises naturally in domains where data is abundant but online interaction is constrained. Examples include modern recommender and advertising systems, which operate on massive logs of user interactions (Afsar et al., 2022; Cai et al., 2017); healthcare and other safety-critical decision problems, which cannot permit exploratory actions (Gottesman et al., 2019; Yu et al., 2021a); robotics and control, which may be limited by real-world cost, wear, and safety (Kober et al., 2013; Zhou et al., 2023; Venkataraman et al., 2025); and autonomous driving and navigation, which rely typically on recorded datasets rather than unrestricted on-policy exploration (Kiran et al., 2022; Shi et al., 2021; Shah et al., 2023).

The core challenge of offline RL is to optimize reward while avoiding actions that are unsupported by the data distribution (Lange et al., 2012; Levine et al., 2020). Offline RL algorithms address this by shaping either the policy or the value-learning objective. Behavior Regularized Actor Critic (BRAC) keeps the learned policy close to the dataset behavior (Fujimoto & Gu, 2021; Tarasov et al., 2023); Conservative Q-Learning (CQL) penalizes high Q-values on unlikely actions to reduce overestimation under dataset shift (Kumar et al., 2020); and implicit methods avoid querying out-of-distribution actions (e.g., IQL) (Kostrikov et al., 2022). Separately, motivated by generative models' success, recent work replaces parametric policies with generative ones to capture multi-modal behavior. These include sequence-modeling policies such as Decision Transformer that predict actions conditioned on return (Chen et al., 2021) and diffusion or flow-based policies such Decision Diffuser (Janner et al., 2022) and Flow Q learning (Park et al., 2025) that generate actions or trajectories by iterative denoising or flow dynamics. For a more detailed discussion on related works, see Appendix A.

Existing offline RL methods typically produce a single policy trained on the offline dataset and then deploy it *as-is* at inference time. These methods do not explicitly exploit inference-time information about the particular states encountered by the agent beyond using it as the input to the pre-trained policy. Recent work on large language models and reasoning suggests that additional inference-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

time computation can improve performance on a given test instance. For example, self-consistency and search-based reasoning methods use extra computation at inference time to refine or select among candidate outputs (Wang et al., 2023a; Yao et al., 2023; Muennighoff et al., 2025), while more recent work has begun to explore reinforcement-learning-based approaches for inference-time improvement (Hou et al., 2025; Zuo et al., 2025). This motivates the question: whether, in offline RL, additional inference-time computation can similarly improve the performance of an agent.

In offline RL, realizing this idea requires identifying what form of inference-time computation can actually improve the deployed policy. An offline learner returns a pre-trained policy π_ψ and an associated critic Q_ϕ, intended to respectively approximate the optimal policy and Q-values. When the policy is deployed without further adaptation, performance depends on how accurate these approximations are in the states and actions *encountered at inference time*. Accurate Q estimation is itself challenging for two reasons. First, the Q-value is intrinsically difficult to estimate because it represents a long-horizon quantity: it aggregates future rewards over many steps starting from a given state-action pair. Second, in offline RL the critic must approximate the Q-value of the *optimal* policy, even though the available data might be generated by a different behavior policy. Beyond the difficulty of Q-estimation, the deployed policy π_ψ is an *amortized* optimizer (Amos, 2023): a single network trained to act well on average over the dataset's states. The action it outputs at a particular state s_t therefore need not be the best action the learned models support there. At deployment, the agent knows the exact state s_t it faces and can spend computation on that state alone rather than on the dataset average. No new data enters this process, so any inference-time gain must come from solving the decision problem at s_t more precisely than the amortized policy π_ψ does. This is possible only if the agent can predict the consequences of acting from s_t, *looking ahead from this specific state* and refining its decision based on what would happen next.

In the Markovian model, the one-step transition dynamics $P(s' \mid s, a)$ and reward function $r(s, a)$ are *local objects*: they depend only on the current state-action pair, rather than on long-horizon estimates, and are independent of whatever policy is being deployed. This fact motivates a different approach: learn world models of P and r from offline data and use these learned functions at inference time to formulate policy. The link between P and r and a corresponding optimal policy is provided by Model Predictive Control (MPC) (Camacho & Bordons, 2007; Rawlings et al., 2017). Conventional MPC optimizes action sequences over rolled-out short-horizon trajectories defined by the function P, thus implicitly generating a policy. We use MPC in a different way, using information gathered from the rollout simulations to do policy optimization, i.e., update the parameters ψ that define our policy π_ψ, and not to merely find the best action sequence. The policy π_ψ is refined at every state s_t the agent visits during deployment. The per-state objective we optimize, an H-step imagined return with a terminal critic, differentiated through the learned model into the policy parameters, has the same form as the objective used to train policies in stochastic value gradients (Heess et al., 2015), MAAC (Clavera et al., 2020), and Dreamer (Hafner et al., 2020). Those methods optimize it during training, averaged over many states; we optimize it at deployment, for the single state the agent is in, using only offline data. See Figure 1 for a visual overview and Section 4 for a detailed description of our method. We now outline our main contributions.

1. We propose *inference-time policy optimization* for offline RL. At every visited state, the agent updates the parameters of its pretrained policy by backpropagating an *imagined* H-step return through a differentiable world model, then acts with the updated policy. The world model consists of a generative next-state sampler, a reward model, and the pretrained critic as a terminal value. This contrasts with prior MPC and planning methods instead optimize action sequences with the policy held fixed or, like POPLIN-P (Wang & Ba, 2020), search over policy parameters online with the cross-entropy method. We call the diffusion-based instantiation `DiffMPC`.
2. We evaluate on D4RL continuous-control benchmarks (Fu et al., 2021) (15 MuJoCo locomotion and 6 AntMaze datasets). `DiffMPC` improves its pretrained ReBRAC policy on all 15 MuJoCo datasets (average normalized score $93.9 \rightarrow 98.5$) and achieves the best score among TD3+BC, IQL, CQL, and ReBRAC on 14 of them. On AntMaze, it improves ReBRAC on 5 of 6 datasets and matches it on the sixth (average $78.0 \rightarrow 85.1$) (see Section 5). We also show that the gain does not come from further optimizing the offline objective: training ReBRAC for a matched number of additional gradient steps raises its MuJoCo average by only 0.4 (Appendix D.7).
3. We compare against model-based and generative offline methods (Decision Transformer, Trajectory Transformer, MOPO, MOReL, MBOP, and Diffuser) and MPC-based methods (D-MPC and TD-MPC2) on the same D4RL tasks. `DiffMPC` achieves the highest average score in every com-

parison: 94.4 versus at most 78.9 for the model-based and generative methods, and 86.9 versus 66.0 for D-MPC. `DiffMPC` also outperforms gradient-based optimization of action sequences (V et al., 2023) on 14 of 15 datasets (Section 5 and Appendix E).

4. Because diffusion unrolling and backpropagation through time (BPTT) dominate per-step cost, we propose `FlowMPC`, which replaces the diffusion sampler with a one-step Mean Flow model. `FlowMPC` also improves zero shot ReBRAC on all 15 MuJoCo datasets, and each world-model call is roughly $8\times$ faster in forward pass and $15\times$ faster including backpropagation (Section 6).

Comparison with existing World Model based Offline RL methods. Although world models have been used in *offline* RL, they are typically leveraged in two ways. Some methods (Kidambi et al., 2020; Yu et al., 2020; 2021b; Ding et al., 2024) use the learned dynamics primarily during *training*, generating imagined rollouts from the offline dataset to construct synthetic experience for policy and value learning. Other methods use a generative model at *inference time* to produce candidates for future optimal state trajectories via sampling (often with guidance via return-conditioning), and then select an action by executing the first action of a sampled plan, without adapting the underlying policy parameters during inference (Ajay et al., 2023; Janner et al., 2022; Ki et al., 2025; Yun et al., 2024). Closest to our design, POPLIN-P (Wang & Ba, 2020) plans in the parameter space of a policy network rather than in action space, but it does so online and searches over parameter perturbations with the cross-entropy method. In contrast, our method operates purely on offline data and takes *gradient steps on the policy parameters* at inference time, backpropagating through a differentiable world model over imagined finite-horizon rollouts.

Comparison with MPC-based methods. Although MPC has been combined with learned dynamics in several recent works, these methods either operate online with environment interaction or perform gradient-free planning over action sequences. Latent-world-model methods such as Dreamer (Hafner et al., 2020; 2021; 2025) and TD-MPC (Hansen et al., 2022; 2024) train jointly with online interaction; the former uses the model only at training time, while the latter performs gradient-free MPPI in the latent space at deployment. Gradient-based and embedded differentiable MPC methods (V et al., 2023; Romero et al., 2026) optimize action sequences or cost-function parameters rather than the policy itself. Diffusion-based MPC methods such as D-MPC (Zhou et al., 2025) sample candidate action sequences from a learned diffusion proposal and select the best by gradient-free random shooting under a multi-step diffusion dynamics model. In contrast, our method operates purely on offline data and uses the current observed state to *adapt the policy parameters at inference time* by backpropagating through a differentiable world model over imagined finite-horizon rollouts. A detailed methodological comparison against each of these is provided in Appendix E.

2 Preliminaries

We consider a Markov decision process (MDP) specified by $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, d_0, r, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $P(s' \mid s, a)$ is the transition kernel, d_0 is the initial state distribution, $r(s, a)$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. A stochastic policy $\pi(a \mid s)$ induces a distribution over trajectories $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$ with density $p^\pi(\tau) = d_0(s_0) \prod_{t \geq 0} \pi(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t)$ and return $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$. The RL objective is to find a return-maximizing policy within a policy class Π, $\pi^\star \in \operatorname{argmax}_{\pi \in \Pi} \mathbb{E}_{\tau \sim p^\pi}[R(\tau)]$.

Temporal-difference learning. TD methods approximate the optimal action-value function $Q^\star(s, a) = \mathbb{E}_{\tau \sim p^{\pi^\star}}[R(\tau) \mid s_0 = s, a_0 = a]$ with a parameterized critic Q_ϕ by minimizing a Bellman error on the data

$$\mathcal{L}_{\mathrm{TD}}(\phi) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(r + \gamma \max_{a' \in \mathcal{A}} Q_{\bar{\phi}}(s', a') - Q_\phi(s, a) \right)^2 \right] ,$$

where $Q_{\bar{\phi}}$ denotes a target network whose parameters $\bar{\phi}$ are a slowly updated copy of ϕ. For continuous action spaces, the maximization is typically implemented via a parameterized actor $\pi_\psi(a \mid s)$, leading to a policy objective of the form $\mathcal{J}(\psi) := \mathbb{E}_{s \sim \mathcal{D},\, a \sim \pi_\psi(\cdot \mid s)}[Q_\phi(s, a)]$.

Offline RL and distribution shift. Offline RL learns a policy from a fixed dataset without additional environment interactions. We assume access to a static dataset of transitions $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^N$ collected by an unknown behavior policy μ. A core difficulty is that naively applying TD learning and policy improvement can drive the learned policy π_ψ toward state-action regions that are weakly represented in $\mathcal{D}$, i.e., the induced discounted state-action occupancy measure d^{π_ψ} moves away from d^μ, where $d^\pi(s, a) := (1 - \gamma) \sum_{t \geq 0} \gamma^t \Pr_{\tau \sim p^\pi}(s_t = s,\, a_t = a)$.

Many offline RL methods address this distribution shift by enforcing an explicit constraint, either at the policy level $D(\pi_\psi(\cdot \mid s) \,\|\, \mu(\cdot \mid s)) \le \varepsilon$, or at the occupancy level $D(d^{\pi_\psi} \,\|\, d^{\mu}) \le \varepsilon$, where D is a divergence or discrepancy measure. Such constrained formulations often introduce additional algorithmic heuristics to obtain stable and competitive performance in practice.

3 Components of the World Model

Our algorithm at inference time requires sampling next state samples conditioned on the current state action pair, and differentiating those samples with respect to the conditioning variables. We refer to any model with this sample plus gradient interface as a differentiable generative world model (DiffGenWM), and in this work we implement it using a conditional generative model. In addition to the DiffGenWM module, our world model includes a differentiable reward model and a terminal value function given by the critic of a pretrained policy. Concretely, we build a world model using the offline dataset of trajectories $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^N$, consisting of (1) a *differentiable diffusion sampler* f_θ to simulate the transition dynamics, (2) a reward model r_ξ to learn the reward of a given state action pair and, (3) pretrained policy π_ψ and the corresponding pretrained critic value Q_ϕ.

1. Differentiable Diffusion Sampler. We learn a parametric, differentiable sampler $s_{t+1} = f_\theta(s_t, a_t, \varepsilon_t),\ \varepsilon_t \sim p_0$, that represents a conditional distribution $p_\theta(s_{t+1} \mid s_t, a_t)$ over next states, with all stochasticity isolated in ε_t via reparameterization. Here, *differentiable* means in (s_t, a_t) for fixed ε_t, not in θ. We train f_θ on the offline dataset $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^N$ as a conditional diffusion model (Ho et al., 2020).

Forward process. Given a horizon K and a schedule $\{\alpha_k\}_{k=1}^K \subset (0,1)$ with $\bar\alpha_k := \prod_{j=1}^k \alpha_j$, we corrupt s_{t+1} via a Gaussian Markov chain whose closed-form marginal at level k is $s_{t+1}^{(k)} = \sqrt{\bar\alpha_k}\, s_{t+1} + \sqrt{1-\bar\alpha_k}\, \epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$.

Reverse process. Conditioned on $c_t := (s_t, a_t)$, we initialize $s_{t+1}^{(K)} = z_K$ with $z_K \sim \mathcal{N}(0, I)$ and iteratively denoise:

$$s_{t+1}^{(k-1)} = \tfrac{1}{\sqrt{\alpha_k}}\Big(s_{t+1}^{(k)} - \tfrac{1-\alpha_k}{\sqrt{1-\bar\alpha_k}}\,\hat\epsilon_\theta(s_{t+1}^{(k)}, k, c_t)\Big) + \sigma_k\, z_{k-1}, \quad z_{k-1} \sim \mathcal{N}(0, I),$$

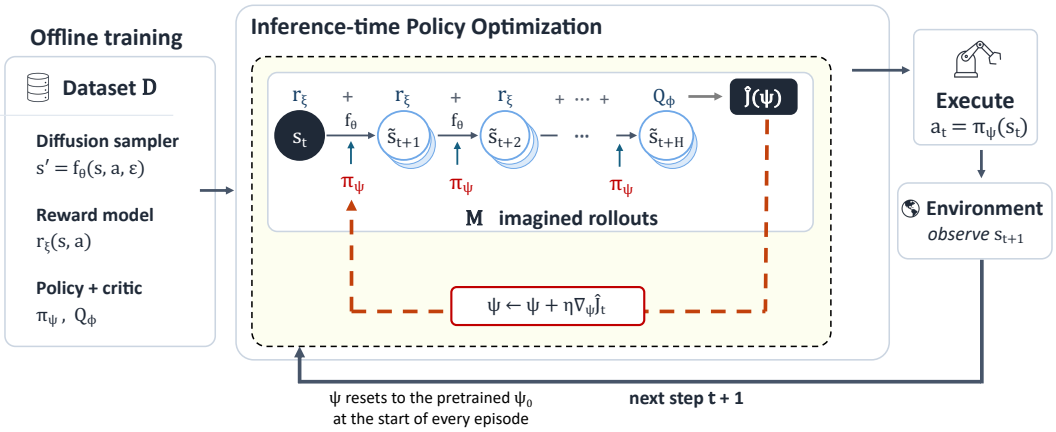
where $\hat\epsilon_\theta$ is a learned noise predictor and $\sigma_k^2 = 1 - \alpha_k$. Collecting the initial draw and the reverse-step Gaussians into $\varepsilon_t := (z_K, z_{K-1}, \ldots, z_0)$, with p_0 the product of $K+1$ standard Gaussians, the sampler $(s_t, a_t) \mapsto s_{t+1}^{(0)}$ is deterministic for fixed ε_t and is therefore differentiable in (s_t, a_t).

Training objective. We minimize the standard conditional denoising loss $\mathcal{L}_{\text{diff}}(\theta) = \mathbb{E}_{(s_t,a_t,s_{t+1})\sim\mathcal{D},\, k\sim\text{Unif}\{1,\ldots,K\},\, \epsilon\sim\mathcal{N}(0,I)}\Big[\|\epsilon - \hat\epsilon_\theta(s_{t+1}^{(k)}, k, c_t)\|_2^2\Big]$, with $s_{t+1}^{(k)}$ given by the forward marginal above. The trained f_θ serves as a learned, differentiable one-step dynamics simulator; see Appendix B.1 for the full construction, including the explicit initialization and reverse-step maps.

2. Reward model. In addition to the transition model, we learn a parametric reward predictor from the same offline data. We parameterize the reward predictor as a function $r_\xi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which maps a state–action pair (s_t, a_t) to a scalar reward estimate. Given an offline dataset of transitions $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^N$, we fit r_ξ by supervised regression on observed rewards, minimizing the loss function $\mathcal{L}_r(\xi) := \mathop{\mathbb{E}}_{(s_t,a_t,r_t)\sim\mathcal{D}}\Big[\big(r_\xi(s_t, a_t) - r_t\big)^2\Big]$. After training, r_ξ serves as a differentiable reward oracle that can be queried at arbitrary (s, a) pairs produced by downstream planning, and can be combined with the learned diffusion transition model to form multi-step return objectives.

3. Policy and Terminal Critic. Given an offline dataset $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^N$, we learn a parametric stochastic policy $\pi_\psi(a \mid s)$ and a parametric action-value function $Q_\phi(s, a)$ via a BC-regularized actor-critic, in the spirit of BRAC (Tarasov et al., 2023; Wu et al., 2019). The critic minimizes a one-step temporal-difference loss $\mathcal{L}_Q(\phi) := \mathbb{E}_{\substack{(s,a,r,s')\sim\mathcal{D}\\ a'\sim\pi_\psi(\cdot|s')}}\Big[\big(Q_\phi(s,a) - r - \gamma Q_{\bar\phi}(s', a')\big)^2\Big]$, where $Q_{\bar\phi}$ is the target network. The actor maximizes value under Q_ϕ while staying close to the dataset behavior via a behavior-cloning term that increases the likelihood of dataset actions under π_ψ, by minimizing $\mathcal{L}_\pi(\psi) := \mathbb{E}_{\substack{(s,a)\sim\mathcal{D}\\ a^\pi\sim\pi_\psi(\cdot|s)}}\Big[-Q_\phi(s, a^\pi) - \lambda \log \pi_\psi(a \mid s)\Big]$, where $\lambda > 0$ controls

Figure 1: Inference-time policy optimization. An offline dataset trains (i) a diffusion sampler f_θ, (ii) a reward model r_ξ, and (iii) a policy π_ψ and terminal critic Q_ϕ. At state s_t, we unroll M imagined rollouts via $\tilde{a}_h = \pi_\psi(\tilde{s}_h)$ and $\tilde{s}_{h+1} = f_\theta(\tilde{s}_h, \tilde{a}_h, \varepsilon_{t+h})$, score them with predicted rewards plus a terminal value, and backpropagate through the rollout to update ψ before executing the first action. For a more detailed forward and backward update schematic diagram see Figure 2.



the strength of regularization. In practice, $Q_{\bar{\phi}}$ is maintained as a slowly updated copy of Q_ϕ to stabilize the bootstrapped target.

4 MPC WITH DIFFERENTIABLE WORLD MODEL

We now describe our receding-horizon model predictive control (MPC) procedure, which uses the learned diffusion world model as a differentiable simulator. The world model is given by the reparameterized transition map $s_{t+1} = f_\theta(s_t, a_t, \varepsilon)$, $\varepsilon \sim p_0$, where f_θ denotes the diffusion sampler of Section 3 unrolled as a computation graph. We combine this dynamics model with the learned reward predictor $r_\xi(s, a)$ and the pretrained critic $Q_\phi(s, a)$. The critic provides a terminal value for truncated rollouts, while r_ξ supplies the reward along imagined trajectories. Without loss of generality, we assume π_ψ is deterministic for the remainder of this section.

Receding-horizon Objective: At a real environment state s_t, MPC optimizes a horizon-H objective over imagined trajectories generated by f_θ. Given a noise sequence $\varepsilon_{t,0:H-1} = (\varepsilon_{t,0}, \ldots, \varepsilon_{t,H-1})$, drawn afresh at every environment step, define the imagined rollout recursively by $\tilde{s}_0 = s_t$, $\tilde{a}_j = \pi_\psi(\tilde{s}_j)$, $\tilde{s}_{j+1} = f_\theta(\tilde{s}_j, \tilde{a}_j, \varepsilon_{t,j})$, where $j = 0, \ldots, H-1$. We evaluate the finite-horizon return by summing predicted rewards and adding a terminal value given by the critic, and define the MPC objective as its expectation over the sampled noise:

$$J_t(\psi) = \mathbb{E}_{\varepsilon_{t,0:H-1} \sim p_0}\Big[\sum_{j=0}^{H-1} \gamma^j \, r_\xi(\tilde{s}_j, \tilde{a}_j) + \gamma^H \, Q_\phi\big(\tilde{s}_H, \pi_\psi(\tilde{s}_H)\big)\Big].$$

Monte Carlo estimate: We approximate $J_t(\psi)$ with M i.i.d. noise sequences $\{\varepsilon^{(m)}_{t,0:H-1}\}_{m=1}^M$, $\varepsilon^{(m)}_{t,0:H-1} \sim p_0$. For each m, generate an imagined rollout $\{\tilde{s}^{(m)}_j, \tilde{a}^{(m)}_j\}_{j=0}^H$:

$$\tilde{s}^{(m)}_0 = s_t, \; \tilde{a}^{(m)}_j = \pi_\psi(\tilde{s}^{(m)}_j), \; \tilde{s}^{(m)}_{j+1} = f_\theta(\tilde{s}^{(m)}_j, \tilde{a}^{(m)}_j, \varepsilon^{(m)}_{t,j}).$$

We split the M particles into M/B disjoint minibatches of size B. For a minibatch $\mathcal{I} \subset \{1, \ldots, M\}$, the estimate is

$$\widehat{J}_{t,\mathcal{I}}(\psi) = \frac{1}{B} \sum_{m \in \mathcal{I}} \Big[\sum_{j=0}^{H-1} \gamma^j \, r_\xi(\tilde{s}^{(m)}_j, \tilde{a}^{(m)}_j) + \gamma^H \, Q_\phi(\tilde{s}^{(m)}_H, \pi_\psi(\tilde{s}^{(m)}_H))\Big].$$

Algorithm 1 Model Predictive Control with a Differentiable World Model

Require: Diffusion sampler f_θ, reward model r_ξ, terminal critic Q_ϕ, pretrained policy ψ_0, horizon H, particles M, minibatch size B, passes E, step size η, discount γ, episode length T.

for each episode **do**
  $\psi \leftarrow \psi_0$ ▷ Reset to pretrained policy
  for $t = 1, 2, \ldots, T$ **do**
    Observe s_t; sample $\varepsilon^{(i)}_{t,0:H-1} \sim p_0$ for $i = 1, \ldots, M$
    for $e = 1, \ldots, E$ and each minibatch $\mathcal{I} \subset \{1, \ldots, M\}$ of size B **do**
      $\tilde{a}^{(i)}_j = \pi_\psi(\tilde{s}^{(i)}_j)$, $\tilde{s}^{(i)}_{j+1} = f_\theta(\tilde{s}^{(i)}_j, \tilde{a}^{(i)}_j, \varepsilon^{(i)}_{t,j})$ from $\tilde{s}^{(i)}_0 = s_t$, $\forall\, i \in \mathcal{I}$ ▷ Imagined rollout
      $\widehat{J} \leftarrow \frac{1}{B} \sum_{i \in \mathcal{I}} \left[\sum_{j=0}^{H-1} \gamma^j \, r_\xi(\tilde{s}^{(i)}_j, \tilde{a}^{(i)}_j) + \gamma^H \, Q_\phi(\tilde{s}^{(i)}_H, \pi_\psi(\tilde{s}^{(i)}_H)) \right]$
      $\psi \leftarrow \psi + \eta \, \nabla_\psi \widehat{J}$ ▷ Policy update
    end for
    Execute $a_t \leftarrow \pi_\psi(s_t)$ and observe (r_t, s_{t+1})
  end for
end for

Gradient-based optimization: At time t, holding f_θ, r_ξ, and Q_ϕ fixed, we make E passes over the minibatches and take one gradient ascent step on each, $\psi \leftarrow \psi + \eta \, \nabla_\psi \widehat{J}_{t,\mathcal{I}}(\psi)$, where $\eta > 0$ is the step size. After these updates, MPC executes $a_t = \pi_\psi(s_t)$ in the real environment, observes (r_t, s_{t+1}), and repeats the procedure at s_{t+1}, starting from the updated ψ. The policy is reset to its pretrained parameters at the start of each episode. See Figure 1 for a visual overview and Algorithm 1 for detailed steps. For a closed-form gradient recursion, see Appendix B.2 and Figure 2 for a detailed flow diagram.

5 Experiments

Datasets. We evaluate our algorithm on the D4RL benchmark (Fu et al., 2021), focusing on continuous-control domains where offline learning and long-horizon credit assignment are challenging. Our main results cover Gym-MuJoCo locomotion datasets (15 tasks spanning medium, expert, medium-expert, medium-replay, and full-replay settings across all three environments HalfCheetah, Hopper, and Walker2d). We also consider all AntMaze datasets (6 tasks spanning umaze, umaze-diverse, medium-play, medium-diverse, large-play, and large-diverse).

Baselines. We compare against strong offline RL baselines that are widely used on D4RL locomotion. These include TD3+BC (Fujimoto & Gu, 2021), which adds behavior cloning regularization to a TD3 actor update; IQL (Kostrikov et al., 2022), which performs offline policy improvement via value-weighted regression without explicit behavior constraints; CQL (Kumar et al., 2020), which trains a conservative critic by penalizing high values on out-of-distribution actions; and ReBRAC (Tarasov et al., 2023), which revisits behavior-regularized actor-critic with practical improvements.

Experimental details. Our approach has an offline training stage and an inference-time adaptation stage. We first train a policy π_ψ and critic Q_ϕ on the offline dataset using ReBRAC (Tarasov et al., 2023). We then train a diffusion transition model f_θ on dataset transitions and a reward predictor r_ξ on dataset rewards using supervised learning. Since the inference-time adaptation differentiates through the diffusion sampler, we refer to this instantiation of our method as `DiffMPC`. Training details and results for both models are deferred to Appendix C. To train the ReBRAC policy we follow the hyper-parameter choices from (Tarasov et al., 2023). We tune the inference time parameters on a single training seed (different from the evaluation seeds). For a detailed description and hyper-parameter choices see Appendix C.

Task Name	ReBRAC	`DiffMPC`
umaze	**99.0** $\pm$ 3.0	**99.0** $\pm$ 3.0
umaze-diverse	89.0 $\pm$ 7.0	**90.0** $\pm$ 6.3
medium-play	86.7 $\pm$ 11.5	**94.4** $\pm$ 6.8
medium-diverse	83.0 $\pm$ 10.1	**94.0** $\pm$ 8.0
large-play	59.0 $\pm$ 25.5	**67.0** $\pm$ 26.1
large-diverse	51.0 $\pm$ 27.0	**66.0** $\pm$ 30.4
Average	77.95	**85.07**

Table 1: Average normalized score on ten unseen seeds on **AntMaze**. $\pm$ represents the standard deviation across seeds.

Dataset	TD3+BC	IQL	CQL	ReBRAC	DiffMPC
halfcheetah					
medium	54.7 ± 0.9	50.0 ± 0.2	46.9 ± 0.4	65.87 ± 1.3	**70.05** ± 1.8
expert	93.4 ± 0.4	95.5 ± 2.1	97.3 ± 1.1	105.22 ± 0.92	**106.14** ± 2.1
medium-expert	89.1 ± 5.6	92.7 ± 2.8	95.0 ± 1.4	100.29 ± 8.2	**105.35** ± 1.8
medium-replay	45.0 ± 1.1	42.1 ± 3.6	45.3 ± 0.3	50.21 ± 0.7	**59.89** ± 1.2
full-replay	75.0 ± 2.5	75.0 ± 0.7	76.9 ± 0.9	82.19 ± 0.9	**85.15** ± 0.9
hopper					
medium	60.9 ± 7.6	65.2 ± 4.2	61.9 ± 6.4	102.05 ± 0.9	**103.38** ± 0.35
expert	**109.6** ± 3.7	108.8 ± 3.1	106.5 ± 9.1	104.73 ± 6.94	105.06 ± 5.02
medium-expert	87.8 ± 10.5	85.5 ± 29.7	96.9 ± 15.1	105.70 ± 8.3	**107.42** ± 5.30
medium-replay	55.1 ± 31.7	89.6 ± 13.2	86.3 ± 7.3	95.82 ± 7.9	**103.14** ± 0.45
full-replay	97.9 ± 17.5	104.4 ± 10.8	101.9 ± 0.6	107.49 ± 0.4	**108.77** ± 0.64
walker2d					
medium	77.7 ± 2.9	80.7 ± 3.4	79.5 ± 3.2	84.78 ± 1.8	**88.91** ± 0.6
expert	110.0 ± 0.6	96.9 ± 32.3	109.3 ± 0.1	112.15 ± 0.2	**116.65** ± 0.41
medium-expert	110.4 ± 0.6	112.1 ± 0.5	109.1 ± 0.2	111.81 ± 0.3	**115.76** ± 1.04
medium-replay	68.0 ± 19.2	75.4 ± 9.3	76.8 ± 10.0	80.67 ± 5.8	**95.87** ± 1.19
full-replay	90.3 ± 5.4	97.5 ± 1.4	94.2 ± 1.9	100.16 ± 8.8	**105.76** ± 2.91
Average	81.7	84.8	85.6	93.9	**98.5**

Table 2: Average normalized score over ten unseen seeds on Gym-MuJoCo tasks. TD3+BC, IQL, and CQL scores are taken from (Tarasov et al., 2023). We ran ReBRAC and DiffMPC with ReBRAC as the pretrained policy. Symbol ± represents standard deviation across seeds.

Dataset	DT	TT	MOPO	MOReL	MBOP	Diffuser	DiffMPC
HalfCheetah							
medium-expert	86.8	95.0	63.3	53.3	**105.9**	88.9 ± 0.3	105.35 ± 1.8
medium	42.6	46.9	42.3	42.1	44.6	42.8 ± 0.3	**70.05** ± 1.8
medium-replay	36.6	41.9	53.1	40.2	42.3	37.7 ± 0.5	**59.89** ± 1.2
Hopper							
medium-expert	107.6	**110.0**	23.7	108.7	55.1	103.3 ± 1.3	107.420 ± 5.30
medium	67.6	61.1	28.0	95.4	48.8	74.3 ± 1.4	**103.38** ± 0.35
medium-replay	82.7	91.5	67.5	93.6	12.4	93.6 ± 0.4	**103.14** ± 0.45
Walker2d							
medium-expert	108.1	101.9	44.6	95.6	70.2	106.9 ± 0.2	**115.76** ± 1.04
medium	74.0	79.0	17.8	77.8	41.0	79.6 ± 0.55	**88.91** ± 0.6
medium-replay	66.6	82.6	39.0	49.8	9.7	70.6 ± 1.6	**95.87** ± 1.19
Average	74.7	78.9	42.1	72.9	47.8	77.5	**94.4**

Table 3: Normalized scores on Gym-MuJoCo. DT, TT, MOPO, MOReL, MBOP, and Diffuser values are taken from Janner et al. (2022). ± represents the standard deviation across the seeds.

Results. Table 2 reports results on Gym-MuJoCo. DiffMPC improves its pretrained ReBRAC policy on all 15 datasets, raising the average normalized score from 93.9 to 98.5, and achieves the best score in the table on 14 of them. In Table 1 we compare DiffMPC with the pre-trained ReBRAC (which outperforms TD3+BC, IQL, CQL and SAC-RND in (Tarasov et al., 2023)) on D4RL Antmaze. DiffMPC raises the average score from 77.95 to 85.07, improving on five of six datasets and matching ReBRAC on umaze; the largest gains are on large-diverse (51.0 → 66.0) and medium-diverse (83.0 → 94.0).

Model-based and generative baselines. We also compare against model-based and generative approaches to offline RL. Decision Transformer (DT) casts it as return-conditioned sequence modeling and predicts actions with a Transformer conditioned on the desired return and history (Chen et al., 2021). Trajectory Transformer (TT) models trajectories as sequences and plans by decoding high-reward trajectories under the learned sequence model (Janner et al., 2021). MOPO learns a dynamics model from offline data and optimizes a policy in the learned model while penalizing rewards by model uncertainty to discourage exploiting model errors (Yu et al., 2020). MOReL constructs a pessimistic surrogate MDP that transitions to an absorbing state where the model is unreliable, then learns a policy within this conservative MDP (Kidambi et al., 2020). MBOP performs model predictive planning in a learned dynamics model, constraining candidate action sequences with a behavior prior and extending the horizon with a value estimate (Argenson & Dulac-Arnold, 2021). Finally, Diffuser learns a diffusion model over trajectories and plans by iterative denoising with flexible conditioning, so that sampling generates high-reward behavior consistent with the constraints (Janner et al., 2022). Since these baselines report results on a subset of D4RL tasks, we use the same subset for direct comparison. Table 3 shows `DiffMPC` achieves the highest average score (94.4 vs. at most 78.9) and the highest average in each of the three environments and is best on 7 of the 9 datasets.

MPC-based baselines. We also compare against recent MPC-based approaches. TD-MPC and TD-MPC2 use learned latent dynamics together with Model Predictive Path Integral (MPPI) control: at each step, candidate action sequences are sampled, rolled out through the latent dynamics, scored using reward and terminal value estimates, and iteratively refined into a sampling distribution from which the first action is executed (Hansen et al., 2022; 2024). These methods are primarily designed for online control, where the replay buffer grows through ongoing environment interaction; Appendix E.1 gives a detailed comparison, and TD-MPC2 trained on the offline datasets scores far below `DiffMPC` on all 15 datasets of Table 2. We further compare against gradient-based optimization of action sequences through learned differentiable rollouts (V et al., 2023), using our world model, critic, and horizon so that only the optimized variable differs. `DiffMPC`, which updates policy parameters instead, scores higher on 14 of 15 datasets (Appendix E.2).

Finally, we compare against D-MPC, a diffusion-based MPC method that trains a diffusion action proposal and a diffusion dynamics model from offline data, then plans by gradient-free random shooting over sampled action sequences (Zhou et al., 2025). Unlike D-MPC, which executes the first action of the best sampled open-loop sequence, our method `DiffMPC` updates the parameters of a pretrained policy by backpropagating through differentiable imagined rollouts inside the MPC loop (see Appendix E.3 for a detailed comparison). D-MPC reports locomotion results only on the medium and medium-replay datasets, so we compare on those six. Table 4 shows that `DiffMPC` achieves an average normalized score of 86.87, compared with 65.98 for D-MPC, and scores higher on all six.

Task Name	D-MPC	`DiffMPC`
halfcheetah medium	46.00 ± 0.17	**70.05** ± 1.8
hopper medium	61.24 ± 2.30	**103.38** ± 0.35
walker2d medium	76.21 ± 2.67	**88.91** ± 0.6
halfcheetah medium-replay	41.12 ± 0.31	**59.89** ± 1.2
hopper medium-replay	92.49 ± 2.23	**103.14** ± 0.45
walker2d medium-replay	78.81 ± 4.19	**95.87** ± 1.19
Average	65.98	**86.87**

Table 4: Normalized scores on the Gym-MuJoCo datasets reported by D-MPC. D-MPC values are taken from Zhou et al. (2025).

Ablation. We conduct extensive ablations on the key hyperparameters of `DiffMPC`: planning horizon H, number of passes E, step size η, DDIM sampling steps K, and number of Monte Carlo particles M, on 17 Gym-MuJoCo datasets (Appendix D). The method is robust across a wide range: moderate horizons ($H \in \{5, 10\}$) and a single pass ($E = 1$) recover most of the gains from inference-time adaptation at modest cost, while M has little effect within $\{512, 1024, 4096\}$. Larger E and η yield further improvements on Walker2d but can amplify model and value errors on termination-sensitive tasks such as Hopper, consistent with known failure modes of offline RL under aggressive policy updates (Kumar et al., 2020; Antos et al., 2007b). The DDIM ablation shows diminishing returns from additional denoising steps on most datasets, motivating the one-step MeanFlow sampler in Section 6. We additionally validate `DiffMPC` with SAC-RND as the pre-trained policy, where it improves the average normalized score from 84.08 to 93.00 (Appendix D.6).

6 Inference-Time Compute and One-Step World Model

H	Zero-shot (ms/step)	DiffMPC (ms/step)
1	0.409 ± 0.053	48.42 ± 0.98
2	0.409 ± 0.053	84.76 ± 1.76
5	0.409 ± 0.053	194.96 ± 14.51

Table 5: Per-step wall-clock time on Hopper while sweeping planning horizon H (DDIM denoising steps $K = 10$, inner gradient steps $E = 1$, one episode of 1000 steps, mean $\pm$ std).

K	Zero-shot (ms/step)	DiffMPC (ms/step)
1	0.390 ± 0.049	57.36 ± 1.51
5	0.390 ± 0.049	121.86 ± 11.37
10	0.390 ± 0.049	198.77 ± 24.64

Table 6: Per-step wall-clock time on Hopper while sweeping DDIM steps K (planning horizon $H = 5$, inner gradient steps $E = 1$, one episode of 1000 steps, mean $\pm$ std).

The main cost of `DiffMPC` is iterative diffusion sampling: every imagined transition requires K sequential denoising steps, and each environment step issues many such transitions. Tables 5 and 6 measure this on hopper-full-replay. The pretrained ReBRAC policy selects an action in about 0.4 ms, whereas `DiffMPC` with $H = 5$ and $K = 10$ takes about 200 ms, roughly $500\times$ longer. Latency grows roughly linearly in both H ($48 \to 195$ ms for $H = 1 \to 5$) and K ($57 \to 199$ ms for $K = 1 \to 10$). Reducing H shortens the lookahead and changes the objective being optimized, whereas reducing K changes only how each transition is sampled, so we target K.

`FlowMPC`: We replace the diffusion sampler with a one-step MeanFlow model (Geng et al., 2025), which maps (s_t, a_t, ϵ_t) to s_{t+1} in a single differentiable forward pass. One-step models are typically less accurate than multi-step samplers, so we train it under a *policy-tilted* objective that upweights dataset transitions whose actions are likely under the pretrained policy. `FlowMPC` improves the average normalized score over the ReBRAC initialization from 94.93 to 97.87, with gains on all 15 datasets. At $K = 10$, the setting used by `DiffMPC`, each world-model call is about $8\times$ faster in the forward pass and $15\times$ faster including backpropagation. A detailed desciption and results are deferred to Appendix F.

Longer horizons. Cost also grows linearly in H because rollouts are autoregressive. A trajectory-level generator that produces $(s_{t+1}, \ldots, s_{t+H})$ in one pass would remove this dependence, but it would have to track a policy that changes at every environment step. Existing trajectory generators either produce synthetic training data guided by the policy being trained (Rigter et al., 2024; Jackson et al., 2024) or produce plans guided by target returns (Ajay et al., 2023); neither is designed for our setting. Policy-gradient guidance during sampling, or periodic fine-tuning of the generator with RL (Black et al., 2024) or adjoint matching (Domingo-Enrich et al., 2025), could close this gap. We view this as a promising path to amortize the per-step cost of long-horizon `DiffMPC`, and leave a thorough exploration to future work.

7 Conclusion

We introduced inference-time policy optimization for offline RL, which refines a pretrained policy at each visited state by backpropagating an imagined return (predicted rewards plus a terminal critic value) through a learned differentiable world model. Our diffusion instantiation, `DiffMPC`, improves its ReBRAC initialization on all 15 D4RL Gym-MuJoCo datasets (93.9 to 98.5) and raises the AntMaze average from 77.95 to 85.07. Since iterative diffusion sampling dominates inference-time cost, we proposed `FlowMPC`, a one-step MeanFlow variant trained under a policy-tilted objective, which also improves ReBRAC on all 15 locomotion datasets while making each world-model call about $8\times$ faster than 10-step diffusion. Together, these results show that inference-time computation, properly amortized, is a practical lever for offline RL. The method still costs more per step than a feedforward policy, and aggressive updates can amplify model errors on termination-sensitive tasks such as Hopper. Towards this, several promising directions remain open: whether trajectory-level world models can generate H-step futures in a single pass while keeping up with a policy that changes at every step, whether the gains carry over to generative base policies such as Flow Q-Learning (Park et al., 2025), and how best to extend the approach to the offline-to-online setting, where the world model can be refined as new data arrives. Another open direction is safe RL: can inference-time policy optimization be extended to respect safety constraints, so that the policy adapts at deployment without increasing the risk of constraint violations?

AI USE STATEMENT

In this work, we used generative AI tools to implement methods, clean and reformat data sets, support qualitative and thematic data analysis, interpret results, and propose or refine hypotheses. We did not use generative AI tools to help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, or design or provide feedback on research methodology or experiments. Generating synthetic data sets and assistance with translation are not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures and to polish the writing of the paper. We have reviewed all AI-assisted work: we checked LLM-generated research ideas for potential plagiarism through a manual literature survey, and LLM-generated code was verified and tested for correctness by one author. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have provided our code to generate our results as a zip file with all the instructions. The hyperparameters are provided in the paper along with in the config files.

REFERENCES

M. Mehdi Afsar, Trafford Crump, and Behrouz Far. Reinforcement learning based recommender systems: A survey. *ACM Comput. Surv.*, 55(7), December 2022. ISSN 0360-0300. doi: 10.1145/3543846. URL https://doi.org/10.1145/3543846.

Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, Tommi S. Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision making? In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=sP1fo2K9DFG.

Brandon Amos. Tutorial on amortized optimization. *Foundations and Trends in Machine Learning*, 16(5):592–732, 06 2023. ISSN 1935-8237. doi: 10.1561/2200000102. URL https://doi.org/10.1561/2200000102.

András Antos, Csaba Szepesvári, and Rémi Munos. Fitted q-iteration in continuous action-space mdps. In J. Platt, D. Koller, Y. Singer, and S. Roweis (eds.), *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007a. URL https://proceedings.neurips.cc/paper_files/paper/2007/file/da0d1111d2dc5d489242e60ebcbaf988-Paper.pdf.

András Antos, Csaba Szepesvári, and Rémi Munos. Fitted q-iteration in continuous action-space mdps. In J. Platt, D. Koller, Y. Singer, and S. Roweis (eds.), *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007b. URL https://proceedings.neurips.cc/paper_files/paper/2007/file/da0d1111d2dc5d489242e60ebcbaf988-Paper.pdf.

Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=OMNB1G5xzd4.

Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=YCWjhGrJFD.

Han Cai, Kan Ren, Weinan Zhang, Kleanthis Malialis, Jun Wang, Yong Yu, and Defeng Guo. Real-time bidding by reinforcement learning in display advertising. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, pp. 661–670, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346757. doi: 10.1145/3018661.3018702. URL https://doi.org/10.1145/3018661.3018702.

Eduardo F. Camacho and Carlos Bordons. *Model Predictive Control*. Springer, London, 2 edition, 2007. doi: 10.1007/978-0-85729-398-5.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=a7APmM4B9d.

Ignasi Clavera, Yao Fu, and Pieter Abbeel. Model-augmented actor-critic: Backpropagating through paths. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Skln2A4YDB.

Marc Peter Deisenroth and Carl Edward Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pp. 465–472, 2011.

Zihan Ding, Amy Zhang, Yuandong Tian, and Qinqing Zheng. Diffusion world model: Future modeling beyond step-by-step rollout for offline reinforcement learning, 2024. URL https://arxiv.org/abs/2402.03570.

Carles Domingo-Enrich, Michal Drozdzal, Brian Karrer, and Ricky T. Q. Chen. Adjoint matching: Fine-tuning flow and diffusion generative models with memoryless stochastic optimal control. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=xQBRrtQM8u.

Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. Rvs: What is essential for offline RL via supervised learning? In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=S874XAIpkR-.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(18):503–556, 2005. URL http://jmlr.org/papers/v6/ernst05a.html.

Rafael Figueiredo Prudencio, Marcos R. O. A. Maximo, and Esther Luna Colombini. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *IEEE Transactions on Neural Networks and Learning Systems*, 35(8):10237–10257, 2024. doi: 10.1109/TNNLS.2023.3250269.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4{rl}: Datasets for deep data-driven reinforcement learning, 2021. URL https://openreview.net/forum?id=px0-N3_KjA.

Scott Fujimoto and Shixiang Gu. A minimalist approach to offline reinforcement learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=Q32U7dzWXpc.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2052–2062. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/fujimoto19a.html.

Zhengyang Geng, Mingyang Deng, Xingjian Bai, Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 75460–75482. Curran Associates, Inc., 2025. doi: 10.52202/085713-2534. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/6d13e085b79d454da5910e4ca82a3d9d-Paper-Conference.pdf.

Omer Gottesman, Fredrik Johansson, Matthieu Komorowski, Aldo Faisal, David Sontag, Finale Doshi-Velez, and Leo Celi. Guidelines for reinforcement learning in healthcare. *Nature Medicine*, 25:16–18, 2019.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 2555–2565, 2019.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations (ICLR)*, 2020.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering Atari with discrete world models. In *International Conference on Learning Representations (ICLR)*, 2021.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025. doi: 10.1038/s41586-025-08744-2.

Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, pp. 8387–8406, 2022.

Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations (ICLR)*, 2024.

Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/148510031349642de5ca0c544f31b2ef-Paper.pdf.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

Zhenyu Hou, Xin Lv, Rui Lu, Jiajie Zhang, Yujiang Li, Zijun Yao, Juanzi Li, Jie Tang, and Yuxiao Dong. T1: Advancing language model reasoning through reinforcement learning and inference scaling. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 23976–24003. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/hou25e.html.

Matthew Thomas Jackson, Michael Matthews, Cong Lu, Benjamin Ellis, Shimon Whiteson, and Jakob Nicolaus Foerster. Policy-guided diffusion. *Reinforcement Learning Journal*, 4:1855–1872, 2024.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 1273–1286, 2021. doi: 10.48550/arXiv.2106.02039. URL https://papers.neurips.cc/paper_files/paper/2021/hash/099fe6b0b444c23836c4a5d07346082b-Abstract.html.

Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9902–9915. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/janner22a.html.

Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 652–661. PMLR, 2016. URL https://proceedings.mlr.press/v48/jiang16.html.

Donghyeon Ki, JunHyeok Oh, Seong-Woong Shim, and Byung-Jun Lee. Prior-guided diffusion planning for offline reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=lC4WKmTScD.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 21810–21823. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/f7efa4f864ae9b88d43527f4b14f750f-Paper.pdf.

B. Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2022.

Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013. doi: 10.1177/0278364913495721. URL https://doi.org/10.1177/0278364913495721.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=68n2s9ZJWF8.

Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/c2073ffa77b5357a498057413bb09d3a-Paper.pdf.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1179–1191. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/0d2b2061826a5df3221116a5085a6052-Paper.pdf.

Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *J. Mach. Learn. Res.*, 4 (null):1107–1149, December 2003. ISSN 1532-4435.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. *Batch Reinforcement Learning*, pp. 45–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-27645-3. doi: 10.1007/978-3-642-27645-3_2. URL https://doi.org/10.1007/978-3-642-27645-3_2.

Jongmin Lee, Wonseok Jeon, Byungjun Lee, Joelle Pineau, and Kee-Eung Kim. Optidice: Offline policy optimization via stationary distribution correction estimation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 6120–6130. PMLR, 2021. URL https://proceedings.mlr.press/v139/lee21f.html.

Sergey Levine, Aviral Kumar, G. Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *ArXiv*, abs/2005.01643, 2020. URL https://api.semanticscholar.org/CorpusID:218486979.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

Xingchao Liu, Chengyue Gong, and qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=XVjTT1nw5z.

Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. Provably good batch off-policy reinforcement learning without great exploration. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1264–1274. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/0dc23b6a0e4abc39904388dd3ffadcd1-Paper.pdf.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025. URL https://aclanthology.org/2025.emnlp-main.1025/.

Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(27):815–857, 2008. URL http://jmlr.org/papers/v9/munos08a.html.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets, 2021. URL https://arxiv.org/abs/2006.09359.

Alexander Nikulin, Vladislav Kurenkov, Denis Tarasov, and Sergey Kolesnikov. Anti-exploration by random network distillation. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 26228–26244. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/nikulin23a.html.

Seohong Park, Qiyang Li, and Sergey Levine. Flow q-learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48104–48127. PMLR, 2025.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning, 2019. URL https://arxiv.org/abs/1910.00177.

James B. Rawlings, David Q. Mayne, and Moritz Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, Madison, WI, 2 edition, 2017. 2nd edition. Available at: https://sites.engineering.ucsb.edu/~jbraw/mpc/.

Martin Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In João Gama, Rui Camacho, Pavel B. Brazdil, Alípio Mário Jorge, and Luís Torgo (eds.), *Machine Learning: ECML 2005*, pp. 317–328, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31692-3.

Marc Rigter, Jun Yamada, and Ingmar Posner. World models via policy-guided trajectory diffusion. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=9CcgO0LhKG.

Angel Romero, Elie Aljalbout, Yunlong Song, and Davide Scaramuzza. Actor–critic model predictive control: Differentiable optimization meets reinforcement learning for agile flight. *IEEE Transactions on Robotics*, 42:673–692, 2026. doi: 10.1109/TRO.2025.3644945.

Dhruv Shah, Arjun Bhorkar, Hrishit Leen, Ilya Kostrikov, Nicholas Rhinehart, and Sergey Levine. Offline reinforcement learning for visual navigation. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 44–54, 2023.

Tianyu Shi, Dong Chen, Kaian Chen, and Zhaojian Li. Offline reinforcement learning for autonomous driving with safety and exploration enhancement, 2021. URL https://arxiv.org/abs/2110.07067.

Denis Tarasov, Vladislav Kurenkov, Alexander Nikulin, and Sergey Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vqGWslLeEw.

Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2139–2148. PMLR, 2016. URL https://proceedings.mlr.press/v48/thomasa16.html.

Jyothir S V, Siddhartha Jalagam, Yann LeCun, and Vlad Sobal. Gradient-based planning with world models, 2023. URL https://arxiv.org/abs/2312.17227.

Sreyas Venkataraman, Yufei Wang, Ziyu Wang, Navin Sriram Ravie, Zackory Erickson, and David Held. Real-world offline reinforcement learning from vision language model feedback. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 13452–13459, 2025. doi: 10.1109/IROS60139.2025.11246918.

Tingwu Wang and Jimmy Ba. Exploring model-based planning with policy networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1exf64KwH.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023a. URL https://openreview.net/forum?id=1PL1NIMMrw.

Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=AHvFDPi-FA.

Ziyu Wang, Alexander Novikov, Konrad Zolna, Josh S Merel, Jost Tobias Springenberg, Scott E Reed, Bobak Shahriari, Noah Siegel, Caglar Gulcehre, Nicolas Heess, and Nando de Freitas. Critic regularized regression. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 7768–7778. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/588cb956d6bbe67078f29f8de420a13d-Paper.pdf.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning, 2019. URL https://arxiv.org/abs/1911.11361.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=5Xc1ecxO1h.

Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement learning in healthcare: A survey. *ACM Comput. Surv.*, 55(1), November 2021a. ISSN 0360-0300. doi: 10.1145/3477600. URL https://doi.org/10.1145/3477600.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 14129–14142. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/a322852ce0df73e204b7e67cbbef0d0a-Paper.pdf.

Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 28954–28967. Curran Associates, Inc., 2021b. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/f29a179746902e331572c483c45e5086-Paper.pdf.

Taeyoung Yun, Sujin Yun, Jaewoo Lee, and Jinkyoo Park. Guided trajectory generation with diffusion models for offline model-based optimization. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 83847–83876. Curran Associates, Inc., 2024. doi: 10.52202/

079017-2665. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/98904db124b2a2463e8c59ec33fc7150-Paper-Conference.pdf.

Gaoyue Zhou, Liyiming Ke, Siddhartha Srinivasa, Abhinav Gupta, Aravind Rajeswaran, and Vikash Kumar. Real world offline reinforcement learning with realistic data source. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7176–7183, 2023. doi: 10.1109/ICRA48891.2023.10161474.

Guangyao Zhou, Sivaramakrishnan Swaminathan, Rajkumar Vasudeva Raju, J Swaroop Guntupalli, Wolfgang Lehrach, Joseph Ortiz, Antoine Dedieu, Miguel Lazaro-Gredilla, and Kevin Patrick Murphy. Diffusion model predictive control. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=pvtgffHtJm.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. Ttrl: Test-time reinforcement learning. In *Advances in Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=VuVhgEiu20.

A Related Works

Offline reinforcement learning. Offline reinforcement learning studies how to learn a high return policy from a fixed dataset while mitigating distribution shift between the learned policy and the logged behavior Lange et al. (2012); Levine et al. (2020). A large family of methods constrains policy improvement to remain close to actions supported by the dataset, either by explicitly regularizing the learned policy toward behavior cloning or by limiting the space of candidate actions used for value based updates Wu et al. (2019); Fujimoto et al. (2019); Kumar et al. (2019); Fujimoto & Gu (2021); Tarasov et al. (2023). Another family focuses on conservative value estimation to reduce overestimation when evaluating actions that are unlikely under the dataset, with CQL penalizing high values assigned to low density actions Kumar et al. (2020). Complementary approaches avoid explicit out of distribution action maximization during policy improvement by extracting policies from value functions using implicit or advantage weighted updates, as in IQL and AWAC Kostrikov et al. (2022); Nair et al. (2021). Across these lines, the shared objective is to maximize return while ensuring that learning and improvement remain grounded in the support of the offline data.

Batch reinforcement learning and approximate dynamic programming. Offline RL is closely related to earlier work on *batch* reinforcement learning and approximate dynamic programming, where policies are learned from logged transition tuples without additional interaction. Classical fitted value-iteration style methods include tree-based fitted Q-iteration (Ernst et al., 2005) and least-squares policy iteration (Lagoudakis & Parr, 2003), along with neural fitted Q-iteration (NFQ) as an early neural approach (Riedmiller, 2005). These methods clarified the basic promise of learning from static data and the practical dependence on function approximation, dataset coverage, and policy evaluation quality; for broader background, see the batch RL chapter by Lange et al. (2012) and the modern offline RL tutorial by Levine et al. (2020).

Coverage, distribution shift, and off-policy evaluation. A recurring theme since early approximate dynamic programming is that learning from a fixed dataset hinges on how well the data covers the state–action regions needed by the learned policy. Analyses of fitted value iteration make this dependence explicit through approximation and sampling error terms that are amplified by distribution shift across Bellman backups (Munos & Szepesvári, 2008; Antos et al., 2007a). In parallel, the offline setting is tightly connected to off-policy evaluation (OPE), which estimates the performance of a target policy from data generated by different behavior policies. Doubly robust estimators (Jiang & Li, 2016) and subsequent refinements aimed at variance control and model-based mixing (Thomas & Brunskill, 2016) highlight the statistical difficulty of evaluating and optimizing policies under mismatch, which later offline RL methods address implicitly through regularization or pessimism.

Deep model-free offline RL. With deep function approximation, naïve off-policy actor–critic can fail offline due to extrapolation error on actions that are weakly supported by the dataset, which is then compounded by bootstrapping (Kumar et al., 2019; Fujimoto et al., 2019). Many practical algorithms can be viewed as enforcing *policy support constraints* during improvement. Behavior-regularized actor–critic (Wu et al., 2019) and advantage-weighted updates (Peng et al., 2019) bias learning toward dataset actions, while critic-regularized regression (Wang et al., 2020) fits policies via weighted regression modulated by value estimates. A particularly influential baseline is TD3+BC, which augments an online actor update with a behavior cloning term and careful normalization (Fujimoto & Gu, 2021). Another line seeks *pessimism* in value estimation: Conservative Q-Learning (CQL) explicitly lowers Q-values for out-of-distribution actions (Kumar et al., 2020), and Implicit Q-Learning (IQL) avoids explicit maximization over unseen actions by combining expectile value regression with advantage-weighted policy updates (Kostrikov et al., 2022). Density-ratio and stationary-distribution correction viewpoints also motivate offline policy optimization; for example, Lee et al. (2021) estimate distribution corrections to optimize policies from fixed data.

Model-based offline RL. Model-based offline RL learns a dynamics model from logged data and uses model rollouts for planning or for generating additional training targets, but must guard against *model exploitation* when rollouts drift out of support. MOPO penalizes uncertainty to discourage out-of-distribution model rollouts (Yu et al., 2020), MOReL constructs pessimistic MDP variants to avoid unreliable regions (Kidambi et al., 2020), and COMBO combines model rollouts with conservative value regularization (Yu et al., 2021b). Planning-first formulations such as MBOP

emphasize using learned models and short-horizon planning as the primary control mechanism under offline training (Argenson & Dulac-Arnold, 2021). Collectively, these works frame a core tradeoff between using model-based lookahead to improve decisions and controlling error accumulation from imperfect learned dynamics.

Offline RL as sequence modeling and generative policy classes. A complementary direction casts offline RL as conditional sequence modeling, learning to output actions from trajectory context rather than performing Bellman backups. Decision Transformer (Chen et al., 2021) conditions on desired returns, while Trajectory Transformer treats trajectories as token sequences for planning and action selection (Janner et al., 2021). More recently, Emmons et al. (2022) study when supervised learning alone can suffice for strong offline performance, emphasizing choices of conditioning information and model capacity. Separately, generative modeling has been used to enlarge policy classes while retaining dataset support; diffusion-based policy representations such as Diffusion-QL (Wang et al., 2023b) and diffusion planners (Janner et al., 2022) demonstrate strong performance but typically require iterative sampling at inference.

B Diffusion Sampler and Rollout Gradients

B.1 Differentiable Diffusion Sampler

Our objective is to learn a parametric differentiable diffusion sampler that conditioned on a given state-action pair s_t, a_t at time t and sampled noise ϵ_t generates the next state s_{t+1} as follows: $s_{t+1} = f_\theta(s_t, a_t, \varepsilon_t), \varepsilon_t \sim p_0(\varepsilon)$. Here f_θ is the *reverse diffusion sampler*, written as a deterministic computation graph, and ε_t is a collection of Gaussian random variables used in the generation process. Equivalently, the diffusion model specifies a conditional distribution over next states, $p_\theta(s_{t+1} \mid s_t, a_t)$, together with a reparameterized sampling procedure $s_{t+1} = f_\theta(s_t, a_t, \varepsilon_t)$ whose randomness is isolated in ε_t.

We learn $p_\theta(s_{t+1} \mid s_t, a_t)$ from an offline dataset of transitions $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^N$. The diffusion model is trained to represent the conditional law of s_{t+1} given (s_t, a_t) by introducing a *forward noising process* on s_{t+1} and a learned *reverse denoising process* that inverts this noising when conditioned on (s_t, a_t). At sampling time, the reverse process induces the map f_θ.

Forward process: Fix a diffusion horizon $K \in \mathbb{N}$ and a variance schedule $\{\alpha_k\}_{k=1}^K \subset (0,1)$. For each transition tuple $(s_t, a_t, s_{t+1}) \in \mathcal{D}$, define a Markovian forward process that progressively corrupts the next state as follows:

$$s_{t+1}^{(0)} = s_{t+1}, \quad s_{t+1}^{(k)} \mid s_{t+1}^{(k-1)} \sim q\big(s^{(k)} \mid s^{(k-1)}\big) = \mathcal{N}\big(\sqrt{\alpha_k}\, s_{t+1}^{(k-1)}, (1-\alpha_k) I\big), k = 1, \dots, K.$$

Let $\bar{\alpha}_k := \prod_{j=1}^k \alpha_j$. This choice implies a closed-form marginal for any noise level k given by $q\big(s_{t+1}^{(k)} \mid s_{t+1}\big) = \mathcal{N}\big(\sqrt{\bar{\alpha}_k}\, s_{t+1}, (1-\bar{\alpha}_k) I\big)$. In particular, one can write a reparameterized sample from the marginal as

$$s_{t+1}^{(k)} = \sqrt{\bar{\alpha}_k}\, s_{t+1} + \sqrt{1-\bar{\alpha}_k}\, \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I).$$

Conditional reverse process: The reverse process aims to sample s_{t+1} conditioned on (s_t, a_t) by iteratively denoising from a Gaussian reference distribution at level K. Let the conditioning be $c_t := (s_t, a_t)$. At sampling time, initialize from a base noise variable $s_{t+1}^{(K)} \sim \mathcal{N}(0, I)$, and apply a learned reverse transition for $k \in [K-1]$ using $s_{t+1}^{(k-1)} = \frac{1}{\sqrt{\alpha_k}}\Big(s_{t+1}^{(k)} - (1-\alpha_k)\, \hat{\epsilon}_\theta\big(s_{t+1}^{(k)}, k, c_t\big)\Big) + \sigma_k z_{k-1}, z_{k-1} \sim \mathcal{N}(0, I)$. Here $\hat{\epsilon}_\theta(\cdot, k, c_t)$ is a parametric predictor of the noise component at level k, and $\{\sigma_k\}$ specifies the reverse-process variance. The final denoised sample is $s_{t+1}^{(0)} \sim p_\theta(\cdot \mid s_t, a_t)$, and we define the reverse sampler f_θ by collecting all Gaussian random variables used by the reverse procedure into $\varepsilon_t := (z_K, z_{K-1}, \dots, z_0)$, so that the sampled next state can be written as $s_{t+1}^{(0)} = f_\theta(s_t, a_t, \varepsilon_t)$. We define the initialization map $g_K : \mathbb{R}^d \to \mathbb{R}^d$ and the reverse-step map $h_k : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^m \times \mathbb{R}^d \to \mathbb{R}^d$ by

$$g_K(z) := z,$$

$$h_k(u, s, a, z) := \frac{1}{\sqrt{\alpha_k}}\Big(u - (1-\alpha_k)\, \hat{\epsilon}_\theta\big(u, k, (s, a)\big)\Big) + \sigma_k z, \qquad \text{for } k = 1, \dots, K-1 \tag{1}$$

so that $s_{t+1}^{(k-1)}=h_k(s_{t+1}^{(k)}, s_t, a_t, z_{k-1})$ and $s_{t+1}^{(K)} = g_K(z_K)$.

Deterministic computation graph for fixed noise. For any fixed realization of ε_t, the mapping $(s_t, a_t) \mapsto s_{t+1}^{(0)}$ is a deterministic composition of (i) linear operations, (ii) evaluations of the denoiser $\hat{\epsilon}_\theta(\cdot, k, c_t)$ at each reverse step, and (iii) additive terms determined by the fixed Gaussian draws. Consequently, f_θ is a differentiable computation graph in its inputs (s_t, a_t) for fixed ε_t.

Learning objective: We train $\hat{\epsilon}_\theta$ to invert the forward corruption of s_{t+1}, conditioned on (s_t, a_t). Using the marginal reparameterization at a randomly chosen diffusion level k, we form $s_{t+1}^{(k)} = \sqrt{\bar{\alpha}_k}\, s_{t+1} + \sqrt{1-\bar{\alpha}_k}\, \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I)$, and minimize the conditional noise-prediction error

$$\mathcal{L}(\theta) = \mathbb{E}_{\substack{(s_t, a_t, s_{t+1}) \sim \mathcal{D} \\ k \sim \mathrm{Unif}(\{1,\dots,K\}) \\ \epsilon \sim \mathcal{N}(0,I)}} \Big[\big\| \epsilon - \hat{\epsilon}_\theta\big(s_{t+1}^{(k)}, k, c_t\big) \big\|_2^2 \Big].$$

Intuitively, this objective teaches the denoiser to recover the injected Gaussian noise at arbitrary noise levels while leveraging (s_t, a_t) as side information. After training, the resulting reverse process defines a conditional generative model $p_\theta(s_{t+1} \mid s_t, a_t)$, and its sampling procedure is precisely the transition map $s_{t+1} = f_\theta(s_t, a_t, \varepsilon_t), \varepsilon_t \sim p_0(\varepsilon)$, which we treat as a learned, differentiable simulator of one-step dynamics.

B.2 Gradient recursion through diffusion rollouts

We provide a schematic diagram of the forward rollout and the backward update in Figure 2. Recall that given a fixed noise sequence $\varepsilon_{t:t+H-1} = (\varepsilon_t, \dots, \varepsilon_{t+H-1})$ define the imagined rollout recursively by

$$\tilde{s}_0 = s_t, \;\; \tilde{a}_j = \pi_\psi(\tilde{s}_j), \;\; \tilde{s}_{j+1} = f_\theta(\tilde{s}_j, \tilde{a}_j, \varepsilon_{t+j}), \tag{2}$$

The next theorem states a compact Jacobian recursion for the gradient of the per-noise objective $L(\psi; \varepsilon_{t:t+H-1})$ under the rollout dynamics in (2). Define the rollout Jacobians, for $j = 0, \dots, H-1$,

$$\begin{aligned}
\Pi_s(j) &:= \nabla_s \pi_\psi(s)\big|_{s=\tilde{s}_j}, \quad \Pi_\psi(j) := \nabla_\psi \pi_\psi(\tilde{s}_j), \\
F_s(j) &:= \nabla_s f_\theta(s, a, \varepsilon_{t+j})\big|_{s=\tilde{s}_j, a=\tilde{a}_j}, \\
F_a(j) &:= \nabla_a f_\theta(s, a, \varepsilon_{t+j})\big|_{s=\tilde{s}_j, a=\tilde{a}_j}.
\end{aligned}$$

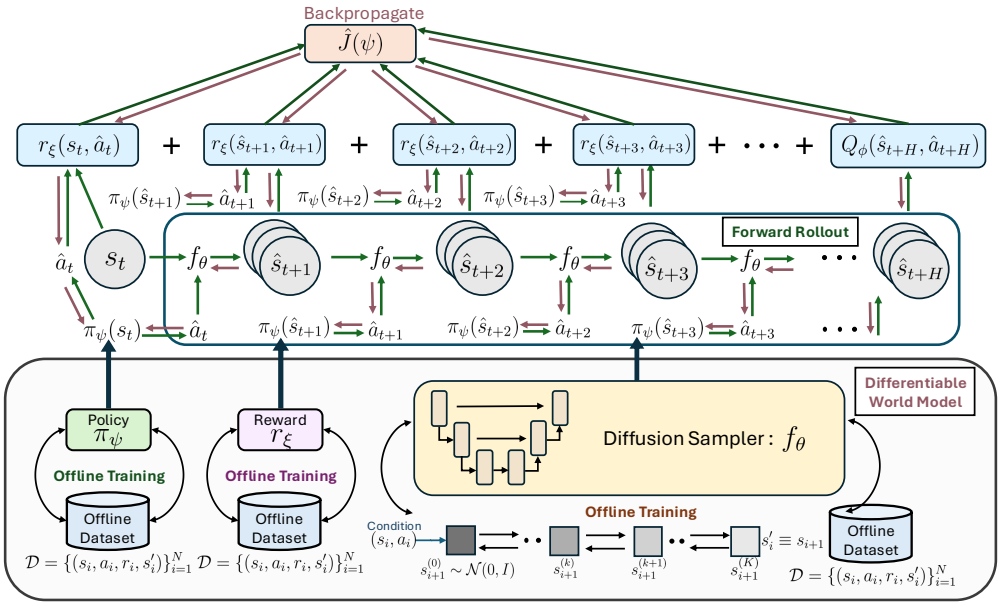
Let $G_j := \nabla_\psi \tilde{s}_j$ and $D_j := \nabla_\psi \tilde{a}_j$. Then $G_0 = 0$ and, for $j = 0, \dots, H-1$,

$$\begin{aligned}
D_j &= \Pi_s(j)\, G_j + \Pi_\psi(j), \\
G_{j+1} &= F_s(j)\, G_j + F_a(j)\, D_j.
\end{aligned}$$

Further define the gradient of the reward and the critic as:

$$\begin{aligned}
r_s(j) &:= \nabla_s r_\xi(\tilde{s}_j, \tilde{a}_j), \;\; r_a(j) := \nabla_a r_\xi(\tilde{s}_j, \tilde{a}_j), \\
Q_s &:= \nabla_s Q_\phi(\tilde{s}_H, \tilde{a}_H), \;\; Q_a := \nabla_a Q_\phi(\tilde{s}_H, \tilde{a}_H).
\end{aligned}$$

Figure 2: Inference-time MPC with a diffusion world model. An offline dataset trains (i) a diffusion sampler f_θ, (ii) a reward model r_ξ, and (iii) a policy π_ψ and terminal critic Q_ϕ. At state s_t, we unroll M imagined rollouts via $\tilde{a}_h = \pi_\psi(\tilde{s}_h)$ and $\tilde{s}_{h+1} = f_\theta(\tilde{s}_h, \tilde{a}_h, \varepsilon_{t+h})$, score them with predicted rewards plus a terminal value, and backpropagate through the rollout to update ψ before executing the first action. Green: forward rollout; Brown: gradient flow.



Theorem B.1 (Gradient recursion) *Fix a time t, a horizon H, and a noise sequence $\varepsilon_{t:t+H-1}$, and let $\{(\tilde{s}_j, \tilde{a}_j)\}_{j=0}^{H}$ be defined by (2). Assume π_ψ is differentiable in ψ and its state input, and f_θ, r_ξ, and Q_ϕ are differentiable in their state and action arguments.*
Then the gradient of the per-noise return is

$$\nabla_\psi L(\psi; \varepsilon_{t:t+H-1}) = \sum_{j=0}^{H-1} \gamma^j \left(r_s(j)\, G_j + r_a(j)\, D_j \right) + \gamma^H \left(Q_s\, G_H + Q_a\, D_H \right).$$

Moreover, if f_θ is implemented by a reverse diffusion recursion as given by (1) then

$$\nabla_a f_\theta(s, a, \varepsilon) = A_0, \qquad \nabla_s f_\theta(s, a, \varepsilon) = B_0,$$

where $A_K = 0$, $B_K = 0$, and for $k = K, \ldots, 1$,

$$A_{k-1} = \frac{\partial h_k}{\partial u} A_k + \frac{\partial h_k}{\partial a}, \qquad B_{k-1} = \frac{\partial h_k}{\partial u} B_k + \frac{\partial h_k}{\partial s},$$

with all partial derivatives evaluated at $(u, s, a, z) = (s^{(k)}, s, a, z_{k-1})$.

Proof.

We fix a time t, horizon H, and a noise sequence $\varepsilon_{t:t+H-1}$. All quantities below are defined along the imagined rollout (2). Since the noise is fixed, the rollout is deterministic, hence the gradient $\nabla_\psi L(\psi; \varepsilon_{t:t+H-1})$ is obtained by repeated application of the chain rule.

1. **Sensitivity recursions:** Recall the rollout recursion (2):

$$\tilde{s}_0 = s_t, \qquad \tilde{a}_h = \pi_\psi(\tilde{s}_h), \qquad \tilde{s}_{h+1} = f_\theta(\tilde{s}_h, \tilde{a}_h, \varepsilon_{t+h}), \qquad h = 0, \ldots, H-1. \quad (3)$$

Define the state and action sensitivities

$$G_h := \nabla_\psi \tilde{s}_h, \qquad D_h := \nabla_\psi \tilde{a}_h. \tag{4}$$

Since $\tilde{s}_0 = s_t$ does not depend on ψ, we have

$$G_0 = 0. \tag{5}$$

For each h, the action is $\tilde{a}_h = \pi_\psi(\tilde{s}_h)$. By the chain rule,

$$D_h = \nabla_\psi \pi_\psi(\tilde{s}_h) + \nabla_s \pi_\psi(s)\big|_{s=\tilde{s}_h} \nabla_\psi \tilde{s}_h. \tag{6}$$

Introduce the policy Jacobians (as in the theorem statement)

$$\Pi_s(h) := \nabla_s \pi_\psi(s)\big|_{s=\tilde{s}_h}, \qquad \Pi_\psi(h) := \nabla_\psi \pi_\psi(\tilde{s}_h). \tag{7}$$

Substituting (7) into (6) yields

$$D_h = \Pi_s(h)\, G_h + \Pi_\psi(h). \tag{8}$$

For each $h \in \{0, \dots, H-1\}$, the next state is $\tilde{s}_{h+1} = f_\theta(\tilde{s}_h, \tilde{a}_h, \varepsilon_{t+h})$. Differentiating with respect to ψ and applying the chain rule gives

$$G_{h+1} = \nabla_s f_\theta(s, a, \varepsilon_{t+h})\big|_{s=\tilde{s}_h, a=\tilde{a}_h} G_h + \nabla_a f_\theta(s, a, \varepsilon_{t+h})\big|_{s=\tilde{s}_h, a=\tilde{a}_h} D_h. \tag{9}$$

Introduce the world-model Jacobians (as in the theorem statement)

$$F_s(h) := \nabla_s f_\theta(s, a, \varepsilon_{t+h})\big|_{s=\tilde{s}_h, a=\tilde{a}_h}, \qquad F_a(h) := \nabla_a f_\theta(s, a, \varepsilon_{t+h})\big|_{s=\tilde{s}_h, a=\tilde{a}_h}. \tag{10}$$

Substituting (10) into (9) yields

$$G_{h+1} = F_s(h)\, G_h + F_a(h)\, D_h. \tag{11}$$

Equations (5), (8), and (11) are exactly the sensitivity recursions stated in Theorem B.1.

2. **Gradient of the per-noise return**

Recall the per-noise return definition:

$$L(\psi; \varepsilon_{t:t+H-1}) = \sum_{h=0}^{H-1} \gamma^h\, r_\xi(\tilde{s}_h, \tilde{a}_h) + \gamma^H\, Q_\phi\Big(\tilde{s}_H, \pi_\psi(\tilde{s}_H)\Big). \tag{12}$$

Fix $h \in \{0, \dots, H-1\}$ and define the stage reward $r_h := r_\xi(\tilde{s}_h, \tilde{a}_h)$. By the chain rule,

$$\nabla_\psi r_h = \nabla_s r_\xi(\tilde{s}_h, \tilde{a}_h)\, G_h + \nabla_a r_\xi(\tilde{s}_h, \tilde{a}_h)\, D_h. \tag{13}$$

Introduce the shorthand

$$r_s(h) := \nabla_s r_\xi(\tilde{s}_h, \tilde{a}_h), \qquad r_a(h) := \nabla_a r_\xi(\tilde{s}_h, \tilde{a}_h). \tag{14}$$

Then (13) becomes

$$\nabla_\psi r_\xi(\tilde{s}_h, \tilde{a}_h) = r_s(h)\, G_h + r_a(h)\, D_h. \tag{15}$$

Define the terminal action $\tilde{a}_H := \pi_\psi(\tilde{s}_H)$ and the terminal value $V_T := Q_\phi(\tilde{s}_H, \tilde{a}_H)$. By the chain rule,

$$\nabla_\psi V_T = \nabla_s Q_\phi(\tilde{s}_H, \tilde{a}_H)\, G_H + \nabla_a Q_\phi(\tilde{s}_H, \tilde{a}_H)\, D_H. \tag{16}$$

Introduce the terminal derivatives

$$Q_s := \nabla_s Q_\phi(\tilde{s}_H, \tilde{a}_H), \qquad Q_a := \nabla_a Q_\phi(\tilde{s}_H, \tilde{a}_H). \tag{17}$$

Then (16) becomes

$$\nabla_\psi Q_\phi(\tilde{s}_H, \tilde{a}_H) = Q_s\, G_H + Q_a\, D_H. \tag{18}$$

Here D_H is defined exactly as in (4), and can be expanded via the same policy sensitivity identity (8) with $h = H$:

$$D_H = \Pi_s(H)\, G_H + \Pi_\psi(H). \tag{19}$$

Differentiating (12) and applying (15) and (18) gives

$$\nabla_\psi L(\psi; \varepsilon_{t:t+H-1}) = \sum_{h=0}^{H-1} \gamma^h \nabla_\psi r_\xi(\tilde{s}_h, \tilde{a}_h) + \gamma^H \nabla_\psi Q_\phi(\tilde{s}_H, \tilde{a}_H)$$
$$= \sum_{h=0}^{H-1} \gamma^h \Big(r_s(h)\, G_h + r_a(h)\, D_h \Big) + \gamma^H \Big(Q_s\, G_H + Q_a\, D_H \Big), \tag{20}$$

which is the gradient expression stated in Theorem B.1.

3. **Jacobian Recursion:** Next we derive and prove the Jacobian recursions for $\nabla_a f_\theta(s, a, \varepsilon)$ and $\nabla_s f_\theta(s, a, \varepsilon)$ under the reverse diffusion implementation (1).
Fix (s, a) and a noise pack $\varepsilon = (z_K, z_{K-1}, \ldots, z_0)$. Let $\{s^{(k)}\}_{k=0}^K$ be generated by

$$s^{(K)} = g_K(z_K), \tag{21}$$
$$s^{(k-1)} = h_k\big(s^{(k)}, s, a, z_{k-1}\big), \qquad k = K, \ldots, 1, \tag{22}$$
$$f_\theta(s, a, \varepsilon) = s^{(0)}. \tag{23}$$

Assume g_K is independent of (s, a) and each $h_k(u, s, a, z)$ is differentiable in (u, s, a). Define the diffusion-depth sensitivities

$$A_k := \nabla_a s^{(k)} \in \mathbb{R}^{d\times m}, \qquad B_k := \nabla_s s^{(k)} \in \mathbb{R}^{d\times d}. \tag{24}$$

Since $s^{(K)} = g_K(z_K)$ and g_K does not depend on (s, a),

$$A_K = 0, \qquad B_K = 0. \tag{25}$$

Fix $k \in \{1, \ldots, K\}$ and write (22) as

$$s^{(k-1)} = h_k(u, s, a, z_{k-1})\Big|_{u=s^{(k)}}. \tag{26}$$

Differentiate (26) with respect to a and apply the chain rule:

$$A_{k-1} = \nabla_a s^{(k-1)} = \frac{\partial h_k}{\partial u} \nabla_a s^{(k)} + \frac{\partial h_k}{\partial a} = \frac{\partial h_k}{\partial u} A_k + \frac{\partial h_k}{\partial a}, \tag{27}$$

where the partial derivatives are evaluated at

$$(u, s, a, z) = \big(s^{(k)},\ s,\ a,\ z_{k-1}\big). \tag{28}$$

Differentiating (26) with respect to s gives

$$B_{k-1} = \nabla_s s^{(k-1)} = \frac{\partial h_k}{\partial u} \nabla_s s^{(k)} + \frac{\partial h_k}{\partial s} = \frac{\partial h_k}{\partial u} B_k + \frac{\partial h_k}{\partial s}, \tag{29}$$

with the same evaluation point (28).

Iterating to obtain $\nabla_a f_\theta$ and $\nabla_s f_\theta$. Starting from (25) and iterating (27)–(29) for $k = K, K-1, \ldots, 1$ yields A_0 and B_0. Since $f_\theta(s, a, \varepsilon) = s^{(0)}$ by (23),

$$\nabla_a f_\theta(s, a, \varepsilon) = \nabla_a s^{(0)} = A_0, \qquad \nabla_s f_\theta(s, a, \varepsilon) = \nabla_s s^{(0)} = B_0. \tag{30}$$

□

C Additional Experimental Details

C.1 World Model

Diffusion Sampler: We train a conditional DDPM dynamics model to predict s_{t+1} from (s_t, a_t) on offline D4RL MuJoCo datasets (Hopper, HalfCheetah, and Walker2d, using standard v2 splits).

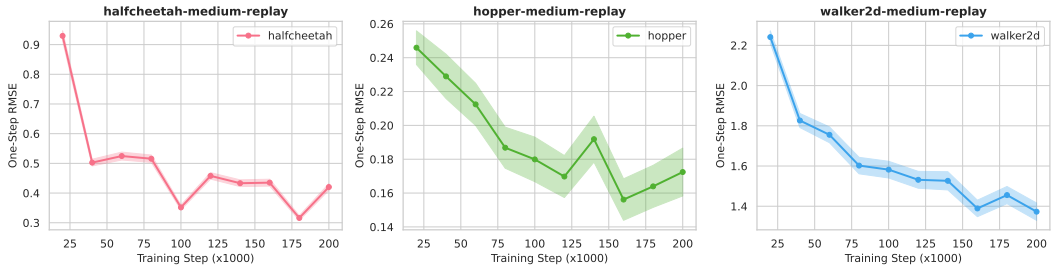


Figure 3: One-step state prediction RMSE of diffusion models across training steps (20k–200k) on medium-replay datasets. RMSE decreases with training for all three environments (halfcheetah, hopper, walker2d), with shaded regions showing standard error over 1000 transitions.

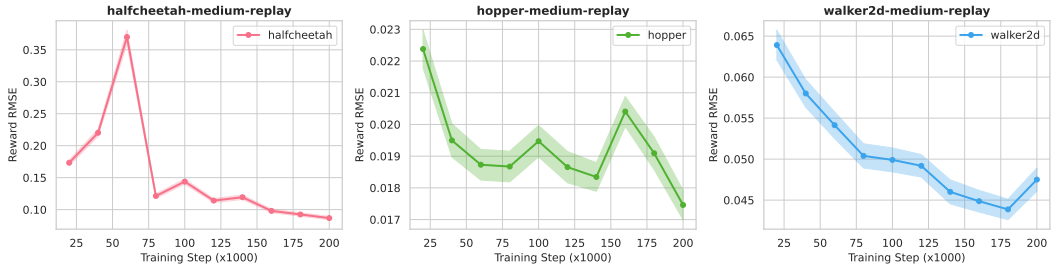


Figure 4: Reward prediction RMSE of reward models across training steps (20k–200k) on medium-replay datasets. All environments show decreasing RMSE with training. Shaded regions indicate standard error over 1000 transitions.

States and next states are standardized with dataset mean/std (with a small ϵ for numerical stability), while actions are left unnormalized, and training minimizes the ϵ-prediction mean-squared error objective. The denoiser is an MLP with time-embedding dimension 128, hidden size 512, and 4 layers, using $T = 200$ diffusion steps and a linear β-schedule from 10^{-4} to 2×10^{-2}. Optimization uses AdamW (learning rate 3×10^{-4}, weight decay 0), batch size 2048, gradient clipping at 1.0, exponential moving average (EMA) of parameters with decay 0.999 and is trained for 200k steps.

Reward Model: We train a reward model $r_\theta(s, a)$ as an MLP on D4RL transitions, where the input is the concatenated state-action vector $[s, a]$ and the output is a scalar immediate reward prediction. The network uses hidden width 512 with 4 layers total (3 hidden layers plus a final linear output layer). States are normalized using dataset statistics while actions are left unnormalized, and training minimizes MSE with AdamW (learning rate 3×10^{-4}, batch size 2048) for 200k steps with gradient clipping at 1.0. We checkpoint every 20k steps and run periodic online validation every 5k steps.

We report the predictive performance of the offline trained diffusion dynamics model and the offline trained reward model across D4RL MuJoCo datasets. Figure 3 shows the diffusion model prediction RMSE across training steps, and Figure 4 reports the reward model prediction RMSE across training steps. For final performance, Table 7 reports both diffusion model accuracy, measured by one-step RMSE, and reward model accuracy, measured by reward prediction RMSE, with uncertainty reported as standard error. For the diffusion model, the one-step RMSE is computed using Monte Carlo samples and normalized by the number of transitions. Overall, both models achieve low error on most datasets.

Policy and Critic Value: ReBRAC is a TD3+BC-style offline RL algorithm that uses three-hidden-layer actor and critic networks with hidden dimension 256 and ReLU activations. The critic applies LayerNorm between hidden layers, and both the actor and critic are trained with behavior-regularization penalties whose coefficients are tuned separately. For D4RL Gym-MuJoCo tasks, ReBRAC uses Adam with batch size 1024, learning rate 10^{-3} for all networks, target update rate $\tau = 5 \times 10^{-3}$, and discount factor $\gamma = 0.99$. The main tuned hyperparameters are the actor and critic regularization coefficients β_1 and β_2. We follow the hyperparameter choices from Tarasov et al. (2023).

Task	Diffusion one-step RMSE ($\pm$ SE)	Reward RMSE ($\pm$ SE)
halfcheetah-random	0.2318 ± 0.000010	0.0477 ± 0.0016
halfcheetah-medium	0.5494 ± 0.000014	0.1292 ± 0.0034
halfcheetah-expert	1.3065 ± 0.000026	0.2763 ± 0.0076
halfcheetah-medium-expert	0.5972 ± 0.000013	0.1622 ± 0.0045
halfcheetah-medium-replay	0.3996 ± 0.000014	0.0918 ± 0.0027
halfcheetah-full-replay	0.4739 ± 0.000014	0.1164 ± 0.0033
hopper-random	0.1268 ± 0.004497	0.0082 ± 0.0003
hopper-medium	0.0784 ± 0.001281	0.0720 ± 0.0030
hopper-expert	0.1375 ± 0.000755	0.1415 ± 0.0058
hopper-medium-expert	0.1678 ± 0.000710	0.0789 ± 0.0033
hopper-medium-replay	0.1008 ± 0.000764	0.0192 ± 0.0005
hopper-full-replay	0.1178 ± 0.002514	0.0208 ± 0.0005
walker2d-random	0.6591 ± 0.009463	0.0244 ± 0.0008
walker2d-medium	1.4068 ± 0.005905	0.0779 ± 0.0020
walker2d-expert	3.3412 ± 0.014321	0.4902 ± 0.0123
walker2d-medium-expert	1.5566 ± 0.026789	0.1060 ± 0.0031
walker2d-medium-replay	1.1100 ± 0.002611	0.0459 ± 0.0013
walker2d-full-replay	1.2560 ± 0.009531	0.0578 ± 0.0022
Average	0.7565 ± 0.000031	0.1092 ± 0.0271

Table 7: Diffusion one-step RMSE and reward prediction RMSE (mean $\pm$ standard error) on D4RL MuJoCo datasets.

C.2 Inference Time Parameters

The inference time parameters for our method are: Horizon Length (H), Inner passes (E), Step size (η), and number of particles M. We tune them on a single different seed, which is different from the 10 evaluation seeds. We tune the planning horizon $H \in \{0, 1, 2, 5, 10\}$, the number of inner passes $E \in \{1, 2, 5\}$, and the number of particles $M \in \{2048, 4096, 8192\}$. For HalfCheetah and Walker2d, we tune the inner step size $\eta \in \{5 \times 10^{-7}, 5 \times 10^{-6}, 5 \times 10^{-5}, 5 \times 10^{-4}\}$ and for Hopper, $\eta \in \{5 \times 10^{-8}, 5 \times 10^{-7}, 5 \times 10^{-6}, 1 \times 10^{-5}, 1 \times 10^{-4}\}$. We keep the DDIM denoising steps $K = 10$. The choices for each dataset is given below in Table 8.

D Ablation Studies

D.1 Horizon Length

We conduct ablation studies to understand the effect of the planning horizon H on the performance of `DiffMPC`. Recall that H controls the number of steps over which imagined rollouts are unrolled before the terminal critic Q_ϕ is applied.

Effect of Planning Horizon H. We sweep $H \in \{1, 2, 5, 10, 20\}$ across all 18 Gym-MuJoCo datasets and report normalized scores in Tables 9–11.

Several consistent patterns emerge across environments and datasets. First, even a short horizon of $H = 1$ or $H = 2$ is sometimes sufficient to recover the gains from inference-time adaptation, suggesting that the immediate one-step gradient signal through the world model provides useful policy improvement. This is particularly visible in the Walker2d family of tasks, where normalized scores improve monotonically or near-monotonically with H (e.g., walker2d-expert improves from 112.94 at $H = 1$ to 113.74 at $H = 20$, and walker2d-medium-expert from 111.71 to 112.77).

Second, moderate horizons ($H = 5$ or $H = 10$) tend to perform well across most datasets without incurring instability. Longer horizons ($H = 20$) occasionally exhibit higher variance or slight degradation, likely due to compounding model errors over longer imagined rollouts. This is most apparent in the hopper-expert task, where the standard deviation grows substantially at $H = 20$ (94.03 ± 18.46) compared to shorter horizons.

Table 8: Hyperparameters for DiffMPC across D4RL datasets. H: horizon length, E: inner passes, η: step size, M: number of particles.

Dataset	H	E	η	M
halfcheetah				
medium	5	5	5×10^{-4}	8,192
expert	2	1	5×10^{-6}	4,096
medium-expert	5	1	5×10^{-5}	8,192
medium-replay	5	5	5×10^{-4}	8,192
full-replay	5	1	5×10^{-4}	4,096
hopper				
medium	2	1	5×10^{-5}	8,192
expert	1	1	5×10^{-8}	8,192
medium-expert	2	1	1×10^{-6}	8,192
medium-replay	5	1	1×10^{-4}	4,096
full-replay	5	1	1×10^{-4}	4,096
walker2d				
medium	5	1	5×10^{-5}	4,096
expert	5	1	5×10^{-5}	8,192
medium-expert	10	1	5×10^{-5}	8,192
medium-replay	5	1	5×10^{-5}	8,192
full-replay	5	1	5×10^{-5}	8,192

Third, on tasks where the pre-trained ReBRAC policy is already near-optimal or the dataset is uninformative (e.g., random datasets such as halfcheetah-random, hopper-random, walker2d-random), inference-time adaptation provides little to no gain across all horizons, consistent with our main results where we observed no improvement on these tasks.

Fourth, on full-replay and medium-replay datasets, where the offline data covers a wider range of behaviors, the gains from increasing H tend to be more stable, as the world model trained on richer data yields more reliable multi-step predictions.

Overall, the ablation confirms that inference-time adaptation is robust across a broad range of horizon choices, with $H \in \{5, 10\}$ offering a good balance between performance and computational cost. Since each additional horizon step requires an extra forward pass through the diffusion sampler and reward model, as well as additional backpropagation through the computation graph, practitioners operating under strict latency constraints may prefer smaller H with minimal loss.

Dataset	$H = 1$	$H = 2$	$H = 5$	$H = 10$	$H = 20$
random	33.66 ± 0.19	33.60 ± 0.32	33.76 ± 0.19	$\mathbf{34.39} \pm 0.23$	34.23 ± 0.34
medium	68.27 ± 0.58	65.52 ± 8.85	$\mathbf{68.97} \pm 0.68$	68.88 ± 0.66	68.77 ± 0.73
medium-replay	51.13 ± 0.85	51.03 ± 0.56	$\mathbf{51.68} \pm 0.66$	51.43 ± 0.94	51.48 ± 0.67
medium-expert	$\mathbf{106.68} \pm 0.84$	106.13 ± 1.31	102.36 ± 10.22	105.79 ± 0.62	106.05 ± 0.43
expert	105.54 ± 0.53	104.87 ± 1.84	106.06 ± 0.36	105.80 ± 0.46	$\mathbf{106.67} \pm 0.36$
full-replay	83.78 ± 0.36	84.40 ± 0.30	$\mathbf{85.15} \pm 0.41$	84.27 ± 0.49	82.51 ± 0.48

Table 9: Effect of planning horizon H on `DiffMPC` for HalfCheetah datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

D.2 Effect of Inner Passes

We next study the effect of the number of inner passes E taken on $\widehat{J}_t(\psi)$ at each real environment state, before executing the next action. Larger E allows more aggressive policy adaptation per step,

Dataset	$H = 1$	$H = 2$	$H = 5$	$H = 10$	$H = 20$
random	9.01 ± 1.14	8.87 ± 1.27	8.99 ± 1.12	9.04 ± 1.22	$\mathbf{9.48} \pm 1.22$
medium-replay	101.68 ± 0.44	101.83 ± 0.31	101.68 ± 0.41	93.44 ± 24.63	$\mathbf{101.96} \pm 0.42$
medium-expert	112.47 ± 1.25	106.48 ± 13.37	112.99 ± 1.07	$\mathbf{113.54} \pm 1.14$	109.38 ± 12.93
full-replay	108.01 ± 0.09	108.02 ± 0.13	108.06 ± 0.12	108.38 ± 0.10	$\mathbf{108.61} \pm 0.09$
expert	107.70 ± 13.12	$\mathbf{110.36} \pm 9.51$	101.86 ± 17.29	98.34 ± 19.36	94.03 ± 18.46

Table 10: Effect of planning horizon H on DiffMPC for Hopper datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

Dataset	$H = 1$	$H = 2$	$H = 5$	$H = 10$	$H = 20$
random	$\mathbf{21.96} \pm 0.01$	$\mathbf{21.96} \pm 0.02$	$\mathbf{21.96} \pm 0.01$	$\mathbf{21.96} \pm 0.02$	$\mathbf{21.96} \pm 0.02$
medium	85.75 ± 0.60	81.83 ± 12.87	83.01 ± 10.20	$\mathbf{86.52} \pm 0.45$	84.98 ± 5.96
medium-replay	76.46 ± 22.93	86.36 ± 7.68	76.35 ± 25.69	$\mathbf{87.99} \pm 3.86$	78.86 ± 18.54
medium-expert	111.71 ± 0.11	111.76 ± 0.09	112.04 ± 0.07	112.30 ± 0.08	$\mathbf{112.77} \pm 0.06$
full-replay	104.00 ± 0.34	104.54 ± 0.76	104.50 ± 0.65	$\mathbf{104.74} \pm 0.54$	104.20 ± 0.29
expert	112.94 ± 0.05	112.93 ± 0.08	113.15 ± 0.04	113.41 ± 0.07	$\mathbf{113.74} \pm 0.08$

Table 11: Effect of planning horizon H on DiffMPC for Walker2d datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

but also increases the computational cost linearly and risks overfitting the policy to the imagined trajectories of a single state s_t. We sweep $E \in \{1, 2, 3, 4, 5\}$. Results are reported in Tables 12–14.

The ablation reveals a clear environment-dependent pattern. On the Walker2d tasks, normalized scores improve monotonically with E in nearly every dataset (e.g., walker2d-expert from 112.99 at $E = 1$ to 114.59 at $E = 5$, walker2d-medium-expert from 111.76 to 113.02, walker2d-medium from 82.27 to 87.56). This indicates that the policy benefits from repeated gradient refinement on the same imagined rollout objective, and that the world model is sufficiently accurate on Walker2d that additional inner steps do not introduce harmful drift. A similar but milder trend appears on the HalfCheetah full-replay and expert datasets.

In contrast, the Hopper environment exhibits markedly different behavior. On hopper-medium-replay, performance degrades sharply and monotonically as E increases, falling from 101.92 at $E = 1$ to 52.78 at $E = 5$, with standard deviations growing to nearly 40. A similar instability appears on hopper-expert, where scores are highest at $E = 1$ and become noisier at larger E. We attribute this to the interaction between repeated policy optimization and Hopper's termination-sensitive locomotion dynamics: Hopper episodes terminate when the agent becomes unhealthy, i.e., when the hopper falls or topples over. Thus, small policy-induced trajectory deviations can produce large return changes. More broadly, repeated optimization against imperfect offline model/value estimates can amplify estimation error and move the policy toward out-of-distribution actions, which is a central failure mode in offline RL (Kumar et al., 2020; Antos et al., 2007b).

On HalfCheetah-random, increasing E also leads to a steady decline in performance ($33.58 \rightarrow 29.75$), which is consistent with the observation from the H ablation that random datasets do not benefit from inference-time adaptation: in the absence of useful policy improvement signal, additional passes simply move the policy away from a reasonable initialization.

Overall, $E = 1$ is a safe default that captures most of the benefit at minimal cost on every environment, while $E \in \{3, 4, 5\}$ can yield additional gains on stable environments such as Walker2d and on some HalfCheetah datasets. These results suggest that the optimal E is environment-specific and should be tuned with awareness of the underlying dynamics' sensitivity to policy perturbations.

D.3 Effect of Inner Step Size

We next study the effect of the inner-loop step size η used in the gradient updates $\psi \leftarrow \psi + \eta \nabla_\psi \widehat{J}_t(\psi)$. A larger η allows the policy to move further per inner step at inference time, which

Dataset	$E = 1$	$E = 2$	$E = 3$	$E = 4$	$E = 5$
random	**33.58** $\pm$ 0.16	32.44 $\pm$ 0.23	31.14 $\pm$ 0.17	30.33 $\pm$ 0.17	29.75 $\pm$ 0.21
medium	**68.89** $\pm$ 0.90	68.74 $\pm$ 0.61	68.51 $\pm$ 0.61	68.34 $\pm$ 0.64	67.98 $\pm$ 0.87
medium-replay	51.20 $\pm$ 0.57	52.22 $\pm$ 0.76	52.14 $\pm$ 0.73	49.99 $\pm$ 5.87	**52.87** $\pm$ 0.61
medium-expert	106.43 $\pm$ 0.49	**106.79** $\pm$ 0.81	106.40 $\pm$ 0.72	106.40 $\pm$ 0.35	106.57 $\pm$ 0.35
expert	105.38 $\pm$ 0.67	105.41 $\pm$ 0.87	105.63 $\pm$ 0.86	**106.00** $\pm$ 0.41	105.65 $\pm$ 0.54
full-replay	84.28 $\pm$ 0.53	85.23 $\pm$ 0.29	85.80 $\pm$ 0.64	85.91 $\pm$ 0.47	**86.00** $\pm$ 0.27

Table 12: Effect of inner passes E on DiffMPC for HalfCheetah datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

Dataset	$E = 1$	$E = 2$	$E = 3$	$E = 4$	$E = 5$
random	8.69 $\pm$ 1.04	8.85 $\pm$ 1.22	9.28 $\pm$ 1.25	8.69 $\pm$ 1.11	**9.72** $\pm$ 1.23
medium-replay	**101.92** $\pm$ 0.28	93.32 $\pm$ 24.50	77.18 $\pm$ 37.39	62.08 $\pm$ 39.72	52.78 $\pm$ 40.01
medium-expert	110.52 $\pm$ 3.70	113.20 $\pm$ 1.24	106.48 $\pm$ 20.95	107.89 $\pm$ 16.97	**113.38** $\pm$ 0.88
full-replay	108.10 $\pm$ 0.12	108.09 $\pm$ 0.11	108.24 $\pm$ 0.09	108.25 $\pm$ 0.14	**108.35** $\pm$ 0.10
expert	**109.74** $\pm$ 6.38	95.33 $\pm$ 17.45	99.35 $\pm$ 19.71	106.80 $\pm$ 13.17	98.08 $\pm$ 18.41

Table 13: Effect of inner passes E on DiffMPC for Hopper datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

Dataset	$E = 1$	$E = 2$	$E = 3$	$E = 4$	$E = 5$
random	**21.97** $\pm$ 0.01	21.96 $\pm$ 0.01	21.96 $\pm$ 0.02	21.96 $\pm$ 0.01	21.96 $\pm$ 0.02
medium	82.27 $\pm$ 13.94	86.05 $\pm$ 0.32	86.93 $\pm$ 0.35	87.35 $\pm$ 0.31	**87.56** $\pm$ 0.33
medium-replay	**89.27** $\pm$ 1.74	80.26 $\pm$ 25.37	85.85 $\pm$ 11.75	66.77 $\pm$ 37.12	81.34 $\pm$ 26.05
medium-expert	111.76 $\pm$ 0.07	112.21 $\pm$ 0.06	112.57 $\pm$ 0.07	112.81 $\pm$ 0.07	**113.02** $\pm$ 0.07
full-replay	104.18 $\pm$ 0.88	104.41 $\pm$ 0.19	104.79 $\pm$ 0.36	**105.24** $\pm$ 0.50	**105.24** $\pm$ 0.54
expert	112.99 $\pm$ 0.11	113.50 $\pm$ 0.08	113.85 $\pm$ 0.09	114.19 $\pm$ 0.12	**114.59** $\pm$ 0.16

Table 14: Effect of inner passes E on DiffMPC for Walker2d datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

can accelerate adaptation but also risks overshooting and pushing the policy into regions where the world model becomes unreliable. We sweep $\eta \in \{5\times10^{-7}, 5\times10^{-6}, 5\times10^{-5}\}$. Results are reported in Tables 15–17.

The ablation reveals a strong environment-dependent pattern that mirrors what we observed for the inner passes E. On the Walker2d tasks, performance improves nearly monotonically with larger η across every non-random dataset (e.g., walker2d-expert from 112.49 at $\eta = 5\times10^{-7}$ to 115.40 at $\eta = 5\times10^{-5}$, walker2d-medium-replay from 59.48 to 79.00, walker2d-medium-expert from 111.42 to 113.17). This indicates that the Walker2d world model is sufficiently accurate that larger passes translate directly into better policies without inducing harmful drift.

On HalfCheetah, the picture is more mixed. Datasets with broad coverage (full-replay, medium-replay) benefit from the largest step size, while expert and random datasets degrade as η grows. The most extreme case is halfcheetah-random, where the score drops from 34.31 at $\eta = 5\times10^{-7}$ to 27.60 at $\eta = 5\times10^{-5}$. Halfcheetah-expert exhibits a similar monotonic decrease (106.70 $\rightarrow$ 103.54), with the largest η also introducing substantial variance.

The Hopper environment again shows the most pronounced instability under aggressive adaptation. On hopper-medium-replay, the largest step size collapses performance from 101.75 at $\eta = 5\times10^{-6}$ to 61.12 at $\eta = 5\times10^{-5}$, with the standard deviation jumping above 40. This parallels the failure mode observed in the E ablation on the same dataset, and reinforces the interpretation that Hopper's narrow stable manifold makes it unusually sensitive to large policy perturbations.

Overall, the optimal η is environment-specific: $\eta = 5\times10^{-5}$ is preferable on Walker2d and on HalfCheetah datasets with broad coverage, while smaller step sizes around $\eta = 5\times10^{-7}$ to 5×10^{-6} are safer on Hopper and on near-optimal policies. A practical default of $\eta = 5\times10^{-6}$ provides a reasonable trade-off, performing close to the best on most datasets while avoiding the catastrophic instability seen at the largest step size on Hopper.

Dataset	$\eta = 5\times10^{-7}$	$\eta = 5\times10^{-6}$	$\eta = 5\times10^{-5}$
random	**34.31** $\pm$ 0.09	33.60 $\pm$ 0.19	27.60 $\pm$ 0.16
medium	68.65 $\pm$ 1.03	**69.03** $\pm$ 0.81	66.55 $\pm$ 0.60
medium-replay	50.96 $\pm$ 0.74	51.38 $\pm$ 0.85	**52.53** $\pm$ 0.62
medium-expert	106.05 $\pm$ 0.25	105.81 $\pm$ 2.41	**106.23** $\pm$ 0.26
expert	**106.70** $\pm$ 0.34	105.66 $\pm$ 0.41	103.54 $\pm$ 8.21
full-replay	82.46 $\pm$ 0.47	84.31 $\pm$ 0.39	**86.04** $\pm$ 0.34

Table 15: Effect of inner step size η on `DiffMPC` for HalfCheetah datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

Dataset	$\eta = 5\times10^{-7}$	$\eta = 5\times10^{-6}$	$\eta = 5\times10^{-5}$
random	**9.00** $\pm$ 1.24	8.45 $\pm$ 0.73	8.40 $\pm$ 0.69
medium-replay	100.35 $\pm$ 3.65	**101.75** $\pm$ 0.52	61.12 $\pm$ 41.11
medium-expert	**112.54** $\pm$ 1.08	112.14 $\pm$ 0.90	112.35 $\pm$ 0.46
full-replay	108.13 $\pm$ 0.13	107.98 $\pm$ 0.10	**108.46** $\pm$ 0.08
expert	99.50 $\pm$ 20.18	98.88 $\pm$ 18.66	**100.53** $\pm$ 18.66

Table 16: Effect of inner step size η on `DiffMPC` for Hopper datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

D.4 Effect of DDIM Sampling Steps

We next study the effect of the number of DDIM sampling steps K used when unrolling the diffusion world model f_θ at inference time. Recall from Section 3 that each call to f_θ requires K sequential reverse-diffusion steps to produce a single next-state sample. Larger K produces samples that more faithfully match the trained conditional distribution $p_\theta(s_{t+1} \mid s_t, a_t)$, but linearly increases both the cost of each forward rollout and the depth of the computation graph through which gradients are backpropagated. We sweep $K \in \{1, 2, 5, 10, 20\}$ across all 18 Gym-MuJoCo datasets, with the

Dataset	$\eta = 5\times10^{-7}$	$\eta = 5\times10^{-6}$	$\eta = 5\times10^{-5}$
random	**21.96** $\pm$ 0.02	**21.96** $\pm$ 0.01	**21.96** $\pm$ 0.01
medium	77.06 $\pm$ 17.66	86.01 $\pm$ 0.46	**87.99** $\pm$ 0.95
medium-replay	59.48 $\pm$ 32.73	71.06 $\pm$ 29.41	**79.00** $\pm$ 27.66
medium-expert	111.42 $\pm$ 0.12	111.74 $\pm$ 0.08	**113.17** $\pm$ 0.12
full-replay	103.97 $\pm$ 0.51	104.60 $\pm$ 1.39	**105.94** $\pm$ 0.71
expert	112.49 $\pm$ 0.06	112.97 $\pm$ 0.08	**115.40** $\pm$ 0.13

Table 17: Effect of inner step size η on DiffMPC for Walker2d datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

remaining hyperparameters fixed at their best per-dataset values from the previous ablations. Results are reported in Tables 18–20.

The picture that emerges from these tables depends on the dataset. On HalfCheetah, the choice of K has a small effect on most datasets: halfcheetah-medium ranges between 68.67 and 69.12, and halfcheetah-medium-replay between 51.19 and 51.56, with differences on the order of one standard deviation. The Walker2d expert and medium-expert tasks behave similarly, varying by only a few tenths of a point across the full range of K. On these datasets, a small number of denoising steps appears to be enough for inference-time optimization, although larger K can still help at the margin.

Other datasets benefit from larger K. Halfcheetah-full-replay improves from 82.90 at $K = 1$ to 84.46 at $K = 20$, and hopper-full-replay and walker2d-full-replay show the same trend. We attribute this to the fact that broader-coverage datasets train more expressive diffusion models whose multi-step denoising procedure captures finer-grained dynamics that a one-step sample may miss.

Walker2d-medium and walker2d-medium-replay go the other way and degrade with larger K. On walker2d-medium, scores fall from 86.09 at $K = 1$ to 81.04 at $K = 20$, with growing variance. We hypothesize that the deeper diffusion computation graph at large K produces noisier or less stable gradient signals during the policy update, partially offsetting the benefits of more accurate samples.

Hopper shows the largest swings across K, particularly on hopper-medium-expert and hopper-expert, where individual K values can shift by several points with large standard deviations. This matches our earlier observations that Hopper is unusually sensitive to perturbations in the inference-time pipeline, and it suggests that K needs to be tuned more carefully here than elsewhere.

Overall, $K \in \{1, 2\}$ is competitive on many datasets at much lower cost, while $K = 10$ tends to work better on the full-replay datasets where extra sampling fidelity helps. This trade-off motivates the one-step MeanFlow sampler in Section 6: where extra denoising steps give diminishing returns, distilling the diffusion process into a single evaluation recovers most of the benefit at a fraction of the cost, and we can fall back to multi-step sampling on the datasets where it actually pays off.

Dataset	$K = 1$	$K = 2$	$K = 5$	$K = 10$	$K = 20$
random	**34.02** $\pm$ 0.20	33.92 $\pm$ 0.17	33.57 $\pm$ 0.22	33.60 $\pm$ 0.20	33.57 $\pm$ 0.08
medium	68.74 $\pm$ 0.56	68.67 $\pm$ 0.81	68.76 $\pm$ 0.49	**69.12** $\pm$ 0.66	68.70 $\pm$ 0.56
medium-replay	51.41 $\pm$ 0.40	51.19 $\pm$ 0.84	**51.56** $\pm$ 0.44	**51.56** $\pm$ 0.97	51.21 $\pm$ 0.96
medium-expert	105.61 $\pm$ 0.18	105.75 $\pm$ 0.33	106.03 $\pm$ 0.84	104.65 $\pm$ 5.25	**106.53** $\pm$ 0.58
expert	104.49 $\pm$ 0.19	104.58 $\pm$ 0.37	**105.53** $\pm$ 0.62	105.19 $\pm$ 0.70	104.67 $\pm$ 2.24
full-replay	82.90 $\pm$ 0.42	82.87 $\pm$ 0.47	84.31 $\pm$ 0.34	84.35 $\pm$ 0.38	**84.46** $\pm$ 0.28

Table 18: Effect of DDIM sampling steps K on DiffMPC for HalfCheetah datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

D.5 Effect of Number of Particles

We finally study the effect of the number of Monte Carlo particles M used to estimate the MPC objective $\widehat{J}_t(\psi)$ at inference time. Recall that M controls the number of i.i.d. noise sequences over which we average the imagined-rollout return; larger M reduces the variance of the gradient

Dataset	$K = 1$	$K = 2$	$K = 5$	$K = 10$	$K = 20$
random	8.58 ± 1.09	$\mathbf{9.32} \pm 1.22$	8.60 ± 1.03	9.21 ± 1.26	8.37 ± 0.82
medium-replay	101.75 ± 0.45	85.73 ± 33.10	101.48 ± 0.51	101.85 ± 0.31	$\mathbf{102.05} \pm 0.45$
medium-expert	113.22 ± 2.39	108.44 ± 14.94	111.06 ± 4.33	108.92 ± 14.62	$\mathbf{113.66} \pm 0.91$
full-replay	107.92 ± 0.17	107.87 ± 0.20	108.04 ± 0.17	108.00 ± 0.09	$\mathbf{108.09} \pm 0.12$
expert	$\mathbf{111.19} \pm 5.36$	99.62 ± 21.08	106.35 ± 12.65	105.80 ± 11.82	100.79 ± 12.62

Table 19: Effect of DDIM sampling steps K on DiffMPC for Hopper datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

Dataset	$K = 1$	$K = 2$	$K = 5$	$K = 10$	$K = 20$
random	21.96 ± 0.01	21.96 ± 0.01	21.96 ± 0.02	$\mathbf{21.97} \pm 0.01$	21.96 ± 0.02
medium	$\mathbf{86.09} \pm 1.13$	85.52 ± 0.99	85.24 ± 5.28	81.95 ± 12.75	81.04 ± 15.06
medium-replay	71.60 ± 32.72	84.44 ± 5.33	$\mathbf{89.03} \pm 2.19$	78.78 ± 22.03	73.33 ± 19.42
medium-expert	111.71 ± 0.07	111.75 ± 0.07	111.74 ± 0.06	$\mathbf{111.77} \pm 0.11$	111.72 ± 0.10
full-replay	103.57 ± 0.35	103.60 ± 0.32	104.24 ± 0.76	104.10 ± 0.57	$\mathbf{104.40} \pm 0.87$
expert	112.71 ± 0.07	112.71 ± 0.08	112.95 ± 0.05	112.97 ± 0.07	$\mathbf{112.98} \pm 0.11$

Table 20: Effect of DDIM sampling steps K on DiffMPC for Walker2d datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

estimator. Since the M particles are independent and processed in parallel, the wall-clock cost of increasing M is largely absorbed by hardware parallelism, and the practically relevant question is not whether larger M is faster but whether DiffMPC is robust across reasonable choices. We sweep $M \in \{512, 1024, 4096\}$ across all 18 Gym-MuJoCo datasets, with the remaining hyperparameters fixed at their best per-dataset values from the previous ablations. Results are reported in Tables 21–23.

The ablation confirms that DiffMPC is stable across this range of M. On nearly every dataset, the difference between the smallest and largest M is well within the noise of a single run: HalfCheetah scores vary by less than one point across the entire range on every dataset, and the Walker2d expert and medium-expert tasks differ by less than 0.5 points across all three M values. This indicates that the gradient estimator is well-behaved at $M = 512$ and that further variance reduction provides only marginal change to the resulting policy.

The takeaway is that M is not a parameter that requires careful tuning within this range. Any choice in $\{512, 1024, 4096\}$ produces qualitatively similar performance, and practitioners can default to whatever value fits comfortably within their available parallel compute budget without expecting significant changes to the final policy.

Dataset	$M = 512$	$M = 1024$	$M = 4096$
random	$\mathbf{34.27} \pm 0.16$	34.22 ± 0.31	33.61 ± 0.14
medium	$\mathbf{68.89} \pm 0.64$	68.72 ± 0.56	68.50 ± 0.75
medium-replay	51.21 ± 0.63	51.19 ± 0.63	$\mathbf{51.69} \pm 0.69$
medium-expert	106.05 ± 0.23	106.11 ± 0.29	$\mathbf{106.88} \pm 0.24$
expert	$\mathbf{106.45} \pm 0.28$	106.09 ± 0.23	105.77 ± 0.39
full-replay	82.65 ± 0.50	82.88 ± 0.52	$\mathbf{84.32} \pm 0.30$

Table 21: Effect of number of particles M on DiffMPC for HalfCheetah datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

Dataset	$M = 512$	$M = 1024$	$M = 4096$
random	8.61 ± 0.83	8.75 ± 1.03	$\mathbf{8.77} \pm 0.94$
medium-replay	$\mathbf{100.22} \pm 4.75$	99.14 ± 10.27	99.90 ± 4.98
medium-expert	110.15 ± 7.17	$\mathbf{112.75} \pm 1.02$	112.74 ± 1.28
full-replay	$\mathbf{108.11} \pm 0.14$	108.05 ± 0.16	108.04 ± 0.14
expert	100.47 ± 16.44	$\mathbf{102.28} \pm 15.09$	101.30 ± 17.34

Table 22: Effect of number of particles M on `DiffMPC` for Hopper datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

Dataset	$M = 512$	$M = 1024$	$M = 4096$
random	$\mathbf{21.97} \pm 0.01$	21.96 ± 0.02	21.96 ± 0.02
medium	85.70 ± 0.44	83.67 ± 5.29	$\mathbf{86.19} \pm 0.68$
medium-replay	77.21 ± 14.71	74.87 ± 24.18	$\mathbf{79.78} \pm 17.78$
medium-expert	111.45 ± 0.09	111.46 ± 0.09	$\mathbf{111.75} \pm 0.07$
full-replay	$\mathbf{103.81} \pm 0.80$	103.58 ± 0.52	103.74 ± 0.61
expert	112.61 ± 0.10	112.68 ± 0.07	$\mathbf{113.03} \pm 0.10$

Table 23: Effect of number of particles M on `DiffMPC` for Walker2d datasets. Normalized scores (mean $\pm$ std) on D4RL. Best score per row is in bold.

D.6 PRETRAINED POLICY: SAC-RND

To test whether DiffMPC depends on the choice of pretrained policy, we replace ReBRAC with SAC-RND (Nikulin et al., 2023). SAC-RND is an anti-exploration method built on soft actor-critic: it trains a random network distillation (RND) predictor on dataset state–action pairs and uses its prediction error as a penalty on out-of-distribution actions in both the actor and critic objectives. We train SAC-RND with the hyperparameters of Nikulin et al. (2023) and apply inference-time policy optimization starting from the resulting policy, using the mean action of the SAC-RND policy as the deterministic policy π_ψ and the minimum of its two critics as the terminal critic Q_ϕ. The diffusion world model and reward model are the same as in the ReBRAC experiments. Table 24 reports normalized scores for a single training seed, averaged over ten evaluation episodes. DiffMPC improves the SAC-RND policy on 14 of the 15 datasets, and raises the average normalized score from 84.06 to 92.99.

Dataset	SAC-RND (pretrained)	DiffMPC
	halfcheetah	
medium	62.90 ± 0.31	**69.40** ± 0.48
expert	46.96 ± 48.83	**64.49** ± 50.78
medium-expert	105.99 ± 0.17	**107.46** ± 0.16
medium-replay	47.78 ± 1.16	**65.95** ± 0.86
full-replay	**82.03** ± 0.42	81.29 ± 0.23
	hopper	
medium	99.70 ± 0.28	**100.83** ± 0.26
expert	110.64 ± 0.31	**110.74** ± 0.34
medium-expert	110.53 ± 0.17	**110.55** ± 0.26
medium-replay	100.51 ± 1.16	**102.21** ± 0.17
full-replay	77.41 ± 17.96	**103.32** ± 14.29
	walker2d	
medium	91.35 ± 2.32	**92.74** ± 0.83
expert	41.50 ± 17.07	**82.03** ± 31.84
medium-expert	94.85 ± 12.08	**108.56** ± 0.12
medium-replay	84.05 ± 2.03	**89.85** ± 1.66
full-replay	104.77 ± 0.10	**105.50** ± 0.13
Average	84.06	**92.99**

Table 24: Normalized score of the pretrained SAC-RND policy and of DiffMPC initialized from it, for a single training seed. Mean $\pm$ standard deviation over ten evaluation episodes.

D.7 ADDITIONAL OFFLINE TRAINING OF THE PRETRAINED POLICY

Since DiffMPC takes additional gradient steps on the policy parameters at inference time, a natural question is whether the same improvement could be obtained by simply training the pretrained policy for longer. To test this, we resume ReBRAC training from the final checkpoint of each seed, using the original objective and hyperparameters, for as many additional gradient steps as DiffMPC performs over a full evaluation episode. DiffMPC performs $E \cdot M/B$ policy updates per environment step (Section 4): 4 on most datasets and 20 on halfcheetah-medium and halfcheetah-medium-replay. Over a 1,000-step episode, this gives the 4,000 and 20,000 additional training steps listed in Table 25. We then evaluate each resulting policy on the same seeds as in Table 2.

Additional training changes the average normalized score by only $+0.42$ ($93.94 \rightarrow 94.36$), and four of the 15 datasets get worse. In contrast, DiffMPC improves every dataset, by $+4.54$ on average, and its gain exceeds that of additional training on 14 of the 15 datasets.

These results are consistent with the pretrained ReBRAC policy being close to convergence after 1M training steps: further optimizing the same dataset-averaged objective barely changes its per-

formance. The same number of gradient updates spent at inference time, each computed at a state the agent actually visits and with the world model predicting the consequences of acting from it, improves performance across the suite.

	Dataset	Extra steps	ReBRAC	+ extra training	Δ	DiffMPC	Δ
halfcheetah	medium	20,000	65.87 ± 1.30	66.08 ± 1.48	+0.21	70.05 ± 1.80	+4.18
	expert	4,000	105.22 ± 0.92	104.78 ± 4.73	−0.44	106.14 ± 2.10	+0.92
	medium-expert	4,000	100.29 ± 8.20	101.24 ± 6.63	+0.95	105.35 ± 1.80	+5.06
	medium-replay	20,000	50.21 ± 0.70	50.99 ± 0.63	+0.78	59.89 ± 1.20	+9.68
	full-replay	4,000	82.19 ± 0.90	82.59 ± 0.85	+0.40	85.15 ± 0.90	+2.96
hopper	medium	4,000	102.05 ± 0.90	102.46 ± 0.40	+0.41	103.38 ± 0.35	+1.33
	expert	4,000	104.73 ± 6.94	104.62 ± 5.13	−0.11	105.06 ± 5.02	+0.33
	medium-expert	4,000	105.70 ± 8.30	108.28 ± 5.17	+2.58	107.42 ± 5.30	+1.72
	medium-replay	4,000	95.82 ± 7.90	94.10 ± 9.77	−1.72	103.14 ± 0.45	+7.32
	full-replay	4,000	107.49 ± 0.40	107.59 ± 0.49	+0.10	108.77 ± 0.64	+1.28
walker2d	medium	4,000	84.78 ± 1.80	85.14 ± 1.83	+0.36	88.91 ± 0.60	+4.13
	expert	4,000	112.15 ± 0.20	112.20 ± 0.21	+0.05	116.65 ± 0.41	+4.50
	medium-expert	4,000	111.81 ± 0.30	111.84 ± 0.38	+0.03	115.76 ± 1.04	+3.95
	medium-replay	4,000	80.67 ± 5.80	80.28 ± 7.91	−0.39	95.87 ± 1.19	+15.20
	full-replay	4,000	100.16 ± 8.80	103.18 ± 0.91	+3.02	105.76 ± 2.91	+5.60
	Average	–	93.94	94.36	+0.42	98.49	+4.54

Table 25: Additional offline training of the pretrained ReBRAC policy versus inference-time policy optimization with DiffMPC. "Extra steps" equals the number of policy updates DiffMPC performs over a 1,000-step episode. Δ is the change relative to ReBRAC. Mean ± standard deviation across seeds; ReBRAC and DiffMPC values are from Table 2.

E Additional Comparison with MPC approaches

E.1 Latent-space world models for continuous control

The use of learned dynamics for policy improvement has a long history. PILCO (Deisenroth & Rasmussen, 2011) fit Gaussian-process dynamics and computed analytic policy gradients via belief-state propagation. Stochastic Value Gradients (Heess et al., 2015) extended this to neural dynamics by reparameterizing stochastic transitions and backpropagating through finite-horizon rollouts. PlaNet (Hafner et al., 2019) introduced a recurrent state-space model and performed cross-entropy planning in the latent space. Two design axes emerge from this literature: *what kind of latent dynamics to learn*, and *how to use it to act*. Two families have come to dominate continuous control along these axes.

Dreamer family. Dreamer (Hafner et al., 2020) couples a recurrent latent dynamics model with an actor and critic trained on imagined latent rollouts; the actor is updated by backpropagating return gradients through the unrolled dynamics. DreamerV2 (Hafner et al., 2021) introduced categorical latents, and DreamerV3 (Hafner et al., 2025) demonstrated robust performance across DMControl, Atari, Crafter, and Minecraft under fixed hyperparameters. The defining feature is *training-time use of the model*: the deployed actor is feedforward and the model is not consulted at inference.

TD-MPC family. TD-MPC (Hansen et al., 2022) takes the opposite stance. It learns a Task-Oriented Latent Dynamics model jointly with reward, value, and a policy prior via TD learning, then performs gradient-free MPPI in the latent space at inference, with the learned policy as a warm-start. The terminal value bootstraps beyond the planning horizon. TD-MPC2 (Hansen et al., 2024) scales this to 104 continuous-control tasks across DMControl, Meta-World, ManiSkill2, and MyoSuite under a single hyperparameter setting. The defining feature is *inference-time planning*: the model is consulted on every step, with substantial per-step compute (TD-MPC2's MPPI predicts ∼9,200 latent states per action) traded for state-conditioned decision making.

Contrast with our work. Our setting differs along three axes, summarized in Table 26. First, *inference-time parameter refinement*: Dreamer optimizes actor parameters at training time and deploys a static actor; TD-MPC optimizes action sequences at inference time but leaves the underlying policy unchanged. We optimize *policy parameters* at inference time via gradient steps through the rollout, so the deployed policy itself adapts in response to the observed state. Second, *offline regime*: both Dreamer and TD-MPC train via environment interaction with jointly learned dynamics, value, and policy components; we operate purely on a fixed offline dataset with a pretrained policy and critic. Third, *generative state-space dynamics*: Dreamer's RSSM and TD-MPC's TOLD are deterministic predictors in compact latent spaces, trained jointly with reward and value; our transition model is a conditional generative model in explicit state space (diffusion, with a one-step MeanFlow variant in Section 6), trained in isolation by maximum-likelihood objectives. The Dreamer-vs-TD-MPC progression is itself evidence that inference-time use of a world model can outperform training-time-only use at the cost of additional deployment compute. We extend this design philosophy to offline RL and replace gradient-free MPPI over action sequences with gradient-based refinement of policy parameters.

	Dreamer family	**TD-MPC family**	**Ours**
When is the model used?	Training	Inference	Inference
What is optimized?	Actor parameters	Action sequences	Policy parameters
How is it optimized?	Gradient (BPTT)	Gradient-free (MPPI)	Gradient (BPTT)
Setting	Online	Online	Offline
Dynamics	Latent (RSSM)	Latent (TOLD)	Explicit (generative)
Deployment cost	Single forward pass	MPPI search	Inner-loop gradient updates

Table 26: Design-space comparison between the Dreamer family, the TD-MPC family, and our approach.

TD-MPC2 outperforms DreamerV3 on average across the 104-task suite, with the largest gains on hard high-dimensional locomotion and manipulation tasks. As TD-MPC2 represents the stronger of the two families and the closest analogue to our inference-time-planning approach, we use it as the representative latent-world-model baseline in our empirical comparison.

Empirical comparison with TD-MPC2. We train TD-MPC2 (Hansen et al., 2024) on each D4RL Gym-MuJoCo dataset using the official implementation, with the offline transitions supplied as the replay buffer. We additionally train an offline-adapted variant of TD-MPC2 in which there is an added behavior-cloning (BC) loss on the policy: besides the usual TD-MPC2 objective, the actor is explicitly trained to match the dataset actions. This extra term pulls updates toward behavior-supported actions, reducing out-of-distribution policy drift during offline training and typically making learning more stable on fixed datasets. Table 27 reports both final and best normalized scores for the two variants. Neither variant approaches the performance of standard offline RL baselines (Section 5): TD-MPC2 is designed for the online sample-efficient regime in which the replay buffer is grown by ongoing environment interaction, and removing this interaction, whether by simply repurposing the online recipe or by an offline adaptation, substantially degrades performance.

E.2 Gradient-based and embedded differentiable MPC

A complementary line of work uses gradient-based optimization through differentiable dynamics inside an MPC loop, either by backpropagating through learned rollouts or by embedding a differentiable MPC solver as a layer in a policy network. We discuss two representative methods.

Gradient-based planning with world models. V et al. (2023) use gradient descent through a learned differentiable world model as an alternative to gradient-free trajectory optimization (CEM, MPPI) for visual model predictive control. At each environment step, they backpropagate the predicted return from an H-step rollout under the learned dynamics back to the action sequence and update the actions via gradient ascent. The dynamics is a Dreamer-style latent world model. They evaluate in the online, sample-efficient regime on visual control benchmarks and report performance on par with or superior to gradient-free MPC and pure policy-based methods. They also propose a

	TD-MPC2		TD-MPC2 (offline with BC loss)	
Dataset	**Final**	**Best**	**Final**	**Best**
halfcheetah-random	4.13 ± 1.17	17.71 ± 0.51	12.75 ± 1.93	16.04 ± 0.77
halfcheetah-medium	0.72 ± 1.56	2.60 ± 0.17	2.88 ± 0.63	3.34 ± 0.74
halfcheetah-medium-replay	7.01 ± 0.64	10.79 ± 0.94	6.41 ± 0.15	13.96 ± 0.31
halfcheetah-medium-expert	1.67 ± 0.74	2.78 ± 0.38	1.35 ± 0.81	3.32 ± 0.72
halfcheetah-expert	-1.71 ± 0.53	-0.40 ± 0.96	-1.79 ± 0.75	0.32 ± 0.22
halfcheetah-full-replay	4.56 ± 0.58	7.68 ± 1.00	6.25 ± 0.51	10.57 ± 0.11
hopper-random	3.66 ± 1.48	8.21 ± 0.30	1.35 ± 0.48	7.86 ± 0.48
hopper-medium	1.32 ± 0.41	1.86 ± 0.40	1.51 ± 0.18	1.97 ± 0.52
hopper-medium-replay	1.80 ± 0.52	3.81 ± 0.47	2.71 ± 0.21	5.17 ± 1.38
hopper-medium-expert	0.78 ± 0.06	1.32 ± 0.35	0.84 ± 0.05	2.35 ± 1.30
hopper-expert	0.72 ± 0.02	1.43 ± 0.49	0.73 ± 0.08	1.73 ± 0.45
hopper-full-replay	4.18 ± 2.35	5.20 ± 1.15	3.77 ± 1.99	7.74 ± 1.02
walker2d-random	2.93 ± 1.96	9.59 ± 4.01	4.82 ± 0.77	7.50 ± 1.37
walker2d-medium	0.55 ± 0.30	8.02 ± 6.28	0.28 ± 0.29	6.35 ± 3.38
walker2d-medium-replay	4.52 ± 1.89	7.93 ± 0.52	2.26 ± 0.75	8.07 ± 0.27
walker2d-medium-expert	-0.11 ± 0.09	0.58 ± 0.23	0.13 ± 0.26	2.42 ± 0.51
walker2d-expert	-0.14 ± 0.02	0.03 ± 0.18	-0.17 ± 0.06	-0.09 ± 0.03
walker2d-full-replay	4.02 ± 1.72	13.40 ± 1.70	5.20 ± 4.15	20.09 ± 2.61

Table 27: TD-MPC2 normalized scores on D4RL Gym-MuJoCo. Mean $\pm$ standard deviation across three seed. Final score is the score at the end of training; best score is the highest score over the course of training.

hybrid variant in which a policy network produces an initial action sequence which is then refined by gradient-based MPC.

Actor-Critic Model Predictive Control. Romero et al. (2026) embed a differentiable MPC solver as the final layer of an actor network in an actor-critic RL framework. The MPC layer uses an *analytic* dynamics model (a quadrotor model) together with a learned cost-map network that maps observations to the MPC's cost-function parameters. Gradients flow end-to-end through the MPC solver into the cost-map network during training, and at deployment the same MPC solver produces actions on every environment step. The method is evaluated on agile quadrotor racing in simulation and on a real platform, and is shown to improve out-of-distribution behavior, robustness to dynamics perturbations, and sample efficiency relative to standard actor-critic baselines.

Comparison with our work. Our setting differs along three axes, summarized in Table 28. First, *what is optimized*: V et al. (2023) optimize an open-loop action sequence at each step (or a policy-initialized refinement of one); Romero et al. (2026) optimize the cost-map network parameters end-to-end through a differentiable MPC layer at training time, with the MPC solver itself producing actions at inference; we optimize the *policy parameters* ψ via gradient steps through the rollout at inference time. Second, *dynamics class*: V et al. (2023) use a learned deterministic latent model, Romero et al. (2026) use known analytic dynamics, and we use a learned conditional generative model in explicit state space (diffusion in our main results, single-step MeanFlow in Section 6). Third, *problem setting*: both prior works operate online; we operate purely on a fixed offline dataset with a pretrained policy and critic, and the inference-time update is the only mechanism by which the deployed policy can improve.

	Jyothir et al.	**Romero et al.**	**Ours**
What is optimized?	Action sequences	Cost-map parameters	Policy parameters
How is it optimized?	Gradient (BPTT)	Gradient through MPC solver	Gradient (BPTT)
When?	Inference	Training and inference	Inference
Dynamics	Learned latent	Analytic	Learned generative
Setting	Online	Online	Offline
Domain	Visual control	Quadrotor flight	D4RL benchmarks

Table 28: Design-space comparison between gradient-based / embedded differentiable MPC methods and our approach.

Dataset	Grad-MPC	DiffMPC
halfcheetah		
medium	68.26 ± 0.63	**69.14** ± 1.17
expert	107.81 ± 0.58	**107.82** ± 0.60
medium-expert	102.41 ± 1.23	**106.28** ± 1.63
medium-replay	51.32 ± 0.60	**63.45** ± 0.59
full-replay	82.37 ± 0.28	**85.69** ± 0.24
hopper		
medium	102.33 ± 0.25	**103.39** ± 0.08
expert	110.20 ± 6.08	**113.06** ± 2.03
medium-expert	**112.87** ± 1.62	110.68 ± 17.30
medium-replay	101.27 ± 0.34	**103.17** ± 0.15
full-replay	107.13 ± 0.14	**108.26** ± 0.10
walker2d		
medium	86.63 ± 0.96	**88.99** ± 0.60
expert	112.54 ± 0.12	**116.53** ± 0.16
medium-expert	112.20 ± 0.05	**117.32** ± 0.73
medium-replay	78.72 ± 19.90	**94.49** ± 0.86
full-replay	104.37 ± 0.93	**106.69** ± 0.75
Average	96.03	**99.66**

Table 29: Average normalized score over the final evaluation and ten episodes on Gym-MuJoCo tasks. The symbol $\pm$ represents the standard deviation across episodes.

Empirical comparison. Among the two prior methods, only V et al. (2023) is directly applicable to our setting: Romero et al. (2026) requires a known analytic dynamics model, which is unavailable on D4RL benchmarks where we only have access to an offline dataset. We therefore compare against an offline adaptation of V et al. (2023), which we refer to as Grad-MPC: at each environment step, we backpropagate the imagined return through our learned diffusion world model to update an open-loop action sequence via gradient ascent, keeping all other components (world model, critic, horizon, number of inner steps) identical to DiffMPC. Table 29 reports the resulting normalized scores. DiffMPC outperforms Grad-MPC on 14 of 15 datasets and improves the average normalized score from 96.03 to 99.66. The gap is most pronounced on datasets with broad coverage and on Walker2d (e.g., walker2d-medium-expert improves from 112.20 to 117.32, halfcheetah-medium-replay from 51.32 to 63.45). We attribute this gap to the difference in what is being optimized: updating policy parameters ψ amortizes the inference-time gradient signal across states and exploits the inductive bias of the pretrained policy, whereas optimizing a fresh open-loop action sequence at every step

discards this structure and is more vulnerable to compounding world-model errors over the planning horizon.

E.3 DIFFUSION-BASED MPC

A closely related thread uses diffusion models for MPC, but with a fundamentally different role for the generative components. We discuss the most prominent recent example.

Diffusion Model Predictive Control. Zhou et al. (2025) propose D-MPC, an online MPC method built on two diffusion models trained on offline data: a multi-step *action proposal* that generates candidate action sequences over a horizon F, and a multi-step *dynamics model* that predicts the corresponding state trajectories. At each environment step, candidate action sequences are sampled from the action proposal, scored by predicted return under the dynamics model, and the best sequence is selected via a sampling-based planner (a variant of random shooting). Only the first action is executed before the procedure repeats. They evaluate on D4RL Gym-MuJoCo and report substantial gains over single-step model-based planning baselines such as MBOP, attributing the improvement to multi-step trajectory-level modeling that mitigates compounding error.

Contrast with our work. D-MPC and our method both use diffusion models inside an MPC loop, but the role of the diffusion model and the optimization mechanism differ along the axes summarized in Table 30. First, *what is optimized*: D-MPC selects the best *action sequence* from candidates sampled by a learned diffusion action proposal; we update the *policy parameters* ψ of a pretrained behavior policy via gradient steps through the rollout. Second, *how it is optimized*: D-MPC is gradient-free, scoring sampled trajectories under the dynamics model; we backpropagate through the entire rollout to obtain gradients with respect to ψ. Third, *dynamics structure*: D-MPC's diffusion model is multi-step and trajectory-level, predicting F future states in a single forward pass; ours is single-step (s_{t+1} given s_t, a_t), with the rollout assembled by repeated application. Fourth, *generative components*: D-MPC trains two diffusion models (action proposal and dynamics); we train a single generative dynamics model and rely on a separately trained behavior policy for actions. For ease of reference, we reproduce the empirical comparison against D-MPC on D4RL datasets from Section 5 in Table 31.

	Zhou et al.	**Ours**
What is optimized?	Action sequence (best candidate)	Policy parameters
How is it optimized?	Gradient-free (random shooting)	Gradient (BPTT)
Dynamics	Multi-step trajectory diffusion	One-step transition diffusion
Action source	Learned diffusion proposal	Pretrained behavior policy

Table 30: Design-space comparison between D-MPC and our approach.

Task Name	D-MPC	`DiffMPC`
halfcheetah medium	46.00 ± 0.17	**70.05** ± 1.8
hopper medium	61.24 ± 2.30	**103.38** ± 0.35
walker2d medium	76.21 ± 2.67	**88.91** ± 0.6
halfcheetah medium-replay	41.12 ± 0.31	**59.89** ± 1.2
hopper medium-replay	92.49 ± 2.23	**103.14** ± 0.45
walker2d medium-replay	78.81 ± 4.19	**95.87** ± 1.19
Average	65.98	**86.87**

Table 31: Average normalized score on ten unseen seeds. $\pm$ represents the standard deviation across seeds.

F FLOWMPC: ONE STEP MODEL

Inference-time policy optimization can improve performance over purely offline-trained policies, but it brings this issue of deployment speed to the forefront. The inner loop requires many model evaluations per environment step, and diffusion-based transition models are especially costly because each simulated transition requires iterative denoising (Ho et al., 2020). As a result, iterative sampling becomes a dominant latency bottleneck in real-time control settings. There is a strong need for faster generative dynamics for differentiable planning.

One way to reduce this deployment cost is to replace iterative diffusion sampling with a *single-step* generative model. Mean-flow (Geng et al., 2025) dynamics provide such a mechanism: the model maps a state-action pair to the next-state prediction in one forward pass, cutting per-transition latency at inference. This is especially attractive in inference-time MPC loops, where each environment step may require many simulated transitions and wall-clock time scales directly with the number of denoising steps. The main drawback is statistical: single-step models are typically less expressive than multi-step diffusion samplers and can incur higher prediction error on complex, multimodal transition distributions, and under rollout these errors compound. To mitigate this, we bias world-model training toward state-action regions that are likely under a strong pretrained policy, so that capacity is spent where the deployed controller will actually query the model.

F.1 MEAN-FLOW MODELS

Mean-flow models (Geng et al., 2025) provide a framework for *one-step* generation by modeling an *average* velocity field, rather than the *instantaneous* velocity used in standard Flow Matching (Lipman et al., 2023). We follow the common continuous-time transport view, where a latent variable $z_t \in \mathbb{R}^d$ evolves over $t \in [0, 1]$ to map between a data distribution and a simple prior. Let $x \sim p_{\text{data}}$ denote a data sample and $\epsilon \sim p_{\text{prior}}$ denote a prior sample (typically $\mathcal{N}(0, I)$). A simple probability path used in rectified-flow style constructions (Liu et al., 2023) is the linear interpolation

$$z_t \;:=\; (1-t)\,x + t\,\epsilon.$$

Along such paths, Flow Matching learns an instantaneous velocity field $v(z_t, t)$ that defines an ODE

$$\frac{d}{dt} z_t \;=\; v(z_t, t),$$

so that generation can be performed by numerically solving the ODE from $t = 1$ to $t = 0$ starting at $z_1 = \epsilon$.

Average velocity. Mean-flow models instead define the *average velocity* between two time points $r < t$ as (Geng et al., 2025)

$$u(z_t, r, t) \;:=\; \frac{1}{t-r} \int_r^t v(z_\tau, \tau)\, d\tau.$$

This quantity is aligned with the displacement along the trajectory: $z_r \;=\; z_t - (t-r)\,u(z_t, r, t)$.

If a neural network u_θ accurately predicts $u(\cdot, r, t)$, then sampling can avoid numerical integration. In particular, *one-step* sampling is obtained by drawing $z_1 = \epsilon \sim p_{\text{prior}}$ and setting

$$z_0 \;=\; z_1 - u_\theta(z_1, 0, 1).$$

MeanFlow Identity and Training Objective. Direct supervision of $u(z_t, r, t)$ is intractable because it is defined by a time integral. Geng et al. (2025) derive an identity relating average and instantaneous velocities:

$$u(z_t, r, t) \;=\; v(z_t, t) - (t-r)\,\frac{d}{dt} u(z_t, r, t),$$

where $\frac{d}{dt}$ is the total derivative along the trajectory. Expanding the total derivative gives

$$\frac{d}{dt} u(z_t, r, t) \;=\; \big\langle \nabla_z u(z_t, r, t),\, v(z_t, t) \big\rangle + \partial_t u(z_t, r, t).$$

This yields a tractable training target by replacing $v(z_t, t)$ with the usual conditional velocity used in Flow Matching. For the linear path above, a standard conditional velocity is $v_{\text{cond}} = \epsilon - x$ (Lipman et al., 2023; Liu et al., 2023). MeanFlow training fits u_θ by regressing to the identity-based target:

$$\mathcal{L}_{\text{MeanFlow}}(\theta) = \mathbb{E}\Big[\big\|u_\theta(z_t, r, t) - \text{stopgrad}\big(v_{\text{cond}} - (t-r)\,\tfrac{d}{dt}u_\theta(z_t, r, t)\big)\big\|_2^2\Big],$$

with expectations over $x \sim p_{\text{data}}$, $\epsilon \sim p_{\text{prior}}$, and a sampling scheme for (r, t) with $0 \le r < t \le 1$. In our world-model setting, x corresponds to the next state (or state increment), and both u_θ and the conditional velocity are parameterized *conditionally* on the current state–action pair.

F.2 One-Step Mean-Flow Dynamics for World Modeling

We learn a conditional, differentiable transition sampler for next-state generation. Given (s_t, a_t), we generate s_{t+1} via a reparameterized map

$$s_{t+1} \;=\; f_\theta(s_t, a_t, \epsilon_t), \qquad \epsilon_t \sim p_0(\epsilon),$$

so that rollout sampling is a single forward pass and gradients with respect to (s_t, a_t) are available.

MeanFlow sampler. We instantiate f_θ using a one-step MeanFlow model (Geng et al., 2025). The model learns an *average-velocity* field $u_\theta(\cdot, r, t\,;\, s, a)$ and uses the MeanFlow sampling $z_r = z_t - (t-r)\,u(z_t, r, t)$. In the one-step case with $(r, t) = (0, 1)$ and $z_1 = \epsilon$, generation is $z_0 = \epsilon - u_\theta(\epsilon, 0, 1)$. We therefore define the conditional transition sampler as

$$f_\theta(s_t, a_t, \epsilon_t) \;:=\; \epsilon_t \;-\; u_\theta(\epsilon_t, 0, 1\,;\, s_t, a_t).$$

Training from offline transitions. We train on $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}$ by treating $x = s_{t+1}$ as the data sample. We draw $\epsilon \sim p_0$ and times $0 \le r < t \le 1$, and form the linear interpolation $z = (1-t)x + t\epsilon$. For this path, the conditional velocity is $v = \epsilon - x$ (Lipman et al., 2023). MeanFlow fits u_θ by enforcing the MeanFlow identity (Geng et al., 2025) through the regression target

$$u_{\text{tgt}} \;=\; v \;-\; (t-r)\,\frac{d}{dt}u_\theta(z, r, t\,;\, s_t, a_t),$$

where the total derivative along the path is $\frac{d}{dt}u_\theta = v\,\partial_z u_\theta + \partial_t u_\theta$ (implemented as a Jacobian-vector product). The training loss is

$$\mathcal{L}_{\text{MF}}(\theta) \;=\; \mathbb{E}\Big[\big\|u_\theta(z, r, t\,;\, s_t, a_t) - \text{sg}\big(u_{\text{tgt}}\big)\big\|_2^2\Big],$$

where $\text{sg}(\cdot)$ denotes stop-gradient.

F.3 Policy-Tilted Flow World Models

A one-step world model trades compute for statistical error: it is fast to sample, but rollout quality depends on where the model is accurate. Since inference-time planning queries transitions induced by the deployed controller, we train the world model under a *tilted* distribution that emphasizes state–action regions likely under a strong pretrained policy $\pi_{\psi_{\text{pre}}}$.

Tilted training objective. Let $\ell_\theta(s, a, s')$ denote the per-transition MeanFlow loss from the previous subsection (the contribution inside $\mathcal{L}_{\text{MF}}(\theta)$) for a transition $(s, a, s') \in \mathcal{D}$. We train the transition model by minimizing the weighted risk

$$\mathcal{L}_{\text{tilt}}(\theta) \;=\; \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\Big[\bar{w}_\eta(s, a)\,\ell_\theta(s, a, s')\Big],$$

where the normalized tilt weight is

$$\bar{w}_\eta(s, a) \;:=\; \frac{\exp\big(\eta\,\log \pi_{\psi_{\text{pre}}}(a \mid s)\big)}{\mathbb{E}_{(s,a)\sim\mathcal{D}}\big[\exp\big(\eta\,\log \pi_{\psi_{\text{pre}}}(a \mid s)\big)\big]},$$

and $\eta \ge 0$ controls the tilt strength (with $\eta = 0$ recovering uniform training). In practice, the denominator is estimated by the minibatch mean so that the average weight in each minibatch is 1.

The weight $\bar{w}_\eta(s, a)$ increases for dataset transitions whose actions have higher likelihood under $\pi_{\psi_{\text{pre}}}(\cdot \mid s)$. This concentrates model capacity on state–action regions that dominate imagined rollouts at deployment, improving rollout fidelity in the MPC loop and mitigating the statistical error introduced by using a one-step MeanFlow transition model.

Task Name	Zero Shot	FlowMPC
halfcheetah-medium	65.42 ± 1.76	**70.06** ± 1.9
halfcheetah-expert	105.33 ± 3.21	**106.96** ± 1.486
halfcheetah-medium-expert	103.79 ± 2.8	**109.89** ± 0.89
halfcheetah-medium-replay	50.78 ± 0.56	**56.51** ± 5.5
halfcheetah-full-replay	82.38 ± 1.32	**85.89** ± 0.95
hopper-medium	102.25 ± 0.23	**103.07** ± 0.65
hopper-expert	100.74 ± 14.9	**101.21** ± 17.2
hopper-medium-expert	107.48 ± 4.5	**107.78** ± 3.06
hopper-medium-replay	98.65 ± 5.15	**102.60** ± 0.59
hopper-full-replay	107.49 ± 0.74	**108.36** ± 0.55
walker2d-medium	84.79 ± 1.52	**88.59** ± 1.2
walker2d-expert	112.19 ± 0.31	**116.44** ± 0.42
walker2d-medium-expert	111.65 ± 0.46	**115.01** ± 1.09
walker2d-medium-replay	83.86 ± 7.38	**90.92** ± 4.2
walker2d-full-replay	101.11 ± 3.9	**104.76** ± 1.12
Average	94.53	**97.87**

Table 32: Average normalized score over the final evaluation and ten unseen training seeds on Gym-MuJoCo tasks, excluding random datasets. The symbol ± represents the standard deviation across the seeds.

F.4 Experiments

Benchmarks and evaluation. We evaluate on the D4RL Gym-MuJoCo locomotion suite (Fu et al., 2021) across HalfCheetah, Hopper, and Walker2d, using the standard v2 datasets in five regimes (medium, expert, medium-expert, medium-replay, full-replay), for a total of 15 tasks. We report D4RL normalized scores computed from episode returns. Each reported value is the mean and standard deviation over 10 evaluation seeds.

Methods compared. ReBRAC is the offline-trained initialization policy (Tarasov et al., 2023) and is the zero shot performance. FlowMPC is our method, which replaces diffusion transition sampling with a one-step MeanFlow transition model and trains the world model with policy-tilted weighting.

Inference-time policy improvement. Both DiffMPC and FlowMPC refine the deployed controller at test time. At each environment step, they generate imagined forward rollouts under a learned differentiable transition model, score trajectories using learned rewards and a terminal critic term, and take a small number of gradient steps through the rollout computation graph before executing the resulting action. The key difference is the transition sampler: DiffMPC uses a conditional diffusion model with K reverse denoising steps per transition, while FlowMPC uses a single-step MeanFlow sampler.

Main results on D4RL. Table 32 summarizes performance on all 15 tasks. FlowMPC improves over the ReBRAC initialization on every dataset, raising the average normalized score from 94.53 to 97.87. Compared with DiffMPC (Table 2), FlowMPC matches or exceeds it on four of the five HalfCheetah datasets and on hopper-medium-expert.

Inference-time compute and latency. Tables 33 and 34 measure the cost of a single transition evaluation and the corresponding forward+backward cost used in inference-time optimization. Across environments, one-step MeanFlow reduces the forward-pass cost by about $8\times$ relative to a $K = 10$ diffusion sampler, with the gap widening linearly as the number of diffusion denoising steps increases. The same trend holds for end-to-end differentiation: MeanFlow reduces total (forward+backward) time by roughly $15\times$ at $K = 10$ and by more than $70\times$ at $K = 50$. These measurements quantify the bottleneck addressed by FlowMPC: rollout generation and differentiation dominate deployment-time compute, and replacing iterative denoising with a one-step sampler yields large, predictable latency reductions.

Diffusion Steps	Diffusion (ms)	MeanFlow (ms)	Speedup
HalfCheetah			
10	1.0200 ± 0.0019	0.1177 ± 0.0017	8.67x
20	2.0184 ± 0.0019	0.1177 ± 0.0017	17.15x
30	3.0852 ± 0.0008	0.1177 ± 0.0017	26.21x
40	4.0280 ± 0.0011	0.1177 ± 0.0017	34.22x
50	5.0487 ± 0.0072	0.1177 ± 0.0017	42.89x
Hopper			
10	1.0211 ± 0.0017	0.1213 ± 0.0018	8.42x
20	2.0214 ± 0.0017	0.1213 ± 0.0018	16.66x
30	3.0788 ± 0.0010	0.1213 ± 0.0018	25.38x
40	4.0237 ± 0.0038	0.1213 ± 0.0018	33.17x
50	5.0508 ± 0.0017	0.1213 ± 0.0018	41.64x
Walker2d			
10	1.0252 ± 0.0017	0.1182 ± 0.0019	8.67x
20	2.0225 ± 0.0018	0.1182 ± 0.0019	17.11x
30	3.0796 ± 0.0016	0.1182 ± 0.0019	26.05x
40	4.0387 ± 0.0027	0.1182 ± 0.0019	34.17x
50	5.0597 ± 0.0007	0.1182 ± 0.0019	42.81x

Table 33: Forward-pass latency comparison between diffusion and MeanFlow on D4RL random datasets (5 runs per measurement). Horizon $H = 1$, batch size $N = 2048$. Action dimension: HalfCheetah $= 6$, Hopper $= 3$, Walker2d $= 6$. MeanFlow uses a single step while diffusion uses the number of denoising steps shown. Values report mean $\pm$ standard deviation in milliseconds.

Diffusion Steps	Diffusion Total (ms)	MeanFlow Total (ms)	Speedup
HalfCheetah			
10	3.2964 ± 0.0011	0.2184 ± 0.0010	15.09x
20	6.5419 ± 0.0019	0.2184 ± 0.0010	29.95x
30	9.8108 ± 0.0016	0.2184 ± 0.0010	44.92x
40	13.0628 ± 0.0009	0.2184 ± 0.0010	59.81x
50	16.3602 ± 0.0011	0.2184 ± 0.0010	74.91x
Hopper			
10	3.3223 ± 0.0012	0.2261 ± 0.0012	14.69x
20	6.5865 ± 0.0003	0.2261 ± 0.0012	29.13x
30	9.8816 ± 0.0019	0.2261 ± 0.0012	43.70x
40	13.1673 ± 0.0006	0.2261 ± 0.0012	58.24x
50	16.4745 ± 0.0021	0.2261 ± 0.0012	72.86x
Walker2d			
10	3.2956 ± 0.0014	0.2182 ± 0.0008	15.10x
20	6.5398 ± 0.0003	0.2182 ± 0.0008	29.97x
30	9.8225 ± 0.0022	0.2182 ± 0.0008	45.02x
40	13.0738 ± 0.0014	0.2182 ± 0.0008	59.92x
50	16.3534 ± 0.0007	0.2182 ± 0.0008	74.95x

Table 34: Total gradient backpropagation time comparison between diffusion and MeanFlow on D4RL random datasets (5 runs). Horizon $H = 1$, batch size $N = 2048$. Action dimension: HalfCheetah $= 6$, Hopper $= 3$, Walker2d $= 6$. MeanFlow uses one step while diffusion uses the number of denoising steps shown. Values report mean $\pm$ standard deviation in milliseconds.