Less Tuning, Better Planning: Simplifying Offline Model-Based Planning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Offline reinforcement learning enables policies to be learned from previously collected experiences without requiring online interaction. However, these policies are typically deployed as fixed, zero-shot agents and lack the ability to adapt their behavior at test time. Offline model-based planning offers a promising way to enable flexible test-time adaptation, but its performance is highly sensitive to critical design choices, particularly the planning horizon and the action proposer. In practice, these choices are often tuned through online evaluation, contradicting the premise of offline RL. In this work, we introduce Soft Horizon AggRegation for Planning (SHARP), an offline plug-and-play planning method that eliminates the need for an online-tuned planning horizon. Instead of using a fixed horizon across all states, SHARP performs soft horizon aggregation by dynamically weighting returns according to model uncertainty estimated from an ensemble of dynamics models. We further investigate the role of the action proposer and find that stronger offline policies do not necessarily lead to better planning performance. Instead, a simple behavior cloning (BC) policy is often sufficient as an action proposer while avoiding the effort required for extensive policy extraction. Combining these insights, we propose SHARP-BC, which consistently outperforms existing baselines while reducing reliance on extensive online hyperparameter tuning.

1 Introduction

Offline reinforcement learning (RL) [1] enables the conversion of previously collected experiences into policies for control problems without requiring online interaction during training. It is particularly appealing in safety-critical or resource-constrained domains where online exploration is expensive, risky, or impractical, such as robotic manipulation [2, 3], autonomous driving [4, 5], and healthcare decision-making [6]. A central challenge in offline RL is extrapolation error: at test time, the learned policy may encounter state-action pairs that are poorly covered by the offline dataset, leading to unreliable value estimates. Prior work typically addresses this challenge by learning policies that favor safe behaviors observed in the dataset [7–10], or by training adaptive policies that condition on histories, or uncertainty estimates to infer environment variation [11, 12]. However, these policies are usually optimized only during training and then deployed as fixed policies in a zero-shot manner, without the ability to adapt their behavior at test time.

To enable adaptive decision-making at deployment, offline model-based planning methods [13, 14] learn dynamic models from offline data and perform model-predictive control [15] during evaluation. A representative line of work follows a plug-and-play paradigm [16, 17], where components learned by offline RL, such as the policy and value function, are used to guide planning at test time. The procedure typically involves (1) generating candidate actions using an action proposer (usually the policy), (2) evaluating their expected returns with the dynamic models and value function, and (3) selecting the best action to be executed in the environment through an optimization procedure. By

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

leveraging knowledge from dynamic models, current plug-and-play methods enable flexible test-time adaptation and effectively improve policy performance [16, 17].

In plug-and-play offline planning, two design choices are especially critical: (1) **planning horizon**: how far the planner should roll out the learned dynamics, and (2) **action proposer**: which action distribution to use to generate candidate trajectories. Existing methods typically expose these components as hyperparameters or design choices, whose best settings can only be identified through online evaluation, contradicting the premise of "offline" RL. In this work, we aim to eliminate such online-tuned choices by designing a planning algorithm that avoids explicit horizon selection and does not require separately tuning the action proposer.

First, we examine the most influential hyperparameters in offline model-based planning: the planning horizon. It controls how far into the future we should roll out, aiming to balance the information gained and errors produced by the dynamic model. Existing methods typically employ a fixed planning horizon across all states and tune it via online evaluation. However, this practice can be highly sensitive and is often impractical. We empirically find that the optimal planning horizon can vary substantially across tasks and even across states, as illustrated in Figure 1. Therefore, using a single fixed horizon may be suboptimal, motivating the need for adaptive horizon selection.

To this end, we propose **Soft Horizon AggRegation for Planning (SHARP)**. Instead of using a fixed horizon across all states, we use soft horizon aggregation to achieve stable performance across various planning horizons. Specifically, we first generate multiple rollout branches from an ensemble of dynamics models, then estimate the mean and variance of the expected returns at every horizon across all branches. The estimated variance is then used to soft-aggregate the return across all horizons: horizons with larger variance receive smaller weights, reflecting greater model uncertainty at those horizons. On D4RL locomotion datasets [18], we show that our method achieves substantially greater stability across different planning horizons. Moreover, it delivers superior performance compared to existing baselines without requiring online hyperparameter tuning.

Second, we investigate the role of the action proposer, an underexplored component in planning. Intuitively, using a higher-performance action proposer should yield better planning performance. Therefore, existing works tune and leverage policies with higher zero-shot performance as their action proposer [16, 17]. However, policy extraction introduces an additional design space that requires online hyperparameter tuning. By conducting a systematic study of different policy extraction methods, including advantage-weighted regression (AWR) [19] and behavior-constrained policy gradients (DDPG+BC) [9], we find that stronger offline policies do not necessarily contribute to better planning performance. Surprisingly, simple **behavior cloning (BC)**, despite its low zero-shot performance, can still serve as a sufficient action proposer to achieve comparable planning performance. Therefore, we suggest using BC as the default action proposer in offline model-based planning, as it reduces the effort required for extensive policy extraction and hyperparameter tuning while maintaining strong planning performance.

Together, we propose **SHARP-BC**, a simple yet effective offline model-based planning framework that combines soft horizon aggregation with a simple BC policy as an action proposer. By eliminating the need for sensitive horizon tuning and reducing reliance on complex policy extraction, our approach simplifies the design of plug-and-play planning methods. Empirically, SHARP-BC achieves competitive or superior performance compared to other existing planning methods, without requiring any extensive online hyperparameter tuning.

2 Preliminaries

This work studies under the framework of Markov decision processes (MDPs), characterized by a tuple $M = (\mathcal{S}, \mathcal{A}, T, r, d_0, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the continuous state and action spaces, $T(s' \mid s, a)$ is the transition probability, $r(s, a)$ is the reward function, d_0 is the initial state distribution, and $\gamma \in [0, 1]$ is the discount factor.

2.1 Offline reinforcement learning

In offline reinforcement learning (offline RL), an agent is given access to a fixed dataset $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^{N}$ collected by some unknown behavior policy, and is not permitted to interact with the environment during training. The objective is to learn a policy $\pi(a \mid s)$ that maximizes the expected discounted return: $J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$.

A common approach in offline RL is to learn a state-action value function (Q-function): $Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t) \mid s_0 = s, a_0 = a\right]$, which estimates the expected return obtained by taking action a in state s and subsequently following policy π. The learned Q-function can then be used for either policy improvement or action evaluation.

2.2 Offline model-based planning

Offline model-based methods decouple policy learning from decision-making by first learning a dynamic model $\hat{T}(s' \mid s,a)$, a reward model $\hat{r}(s,a)$, and often a value function $\hat{Q}(s,a)$ from the offline dataset. Then, these methods leverage these components to perform decision-making at test time, rather than directly executing the policy, by simulating candidate trajectories, evaluating them using the learned models, and selecting the most promising actions according to a planning objective.

Offline model-based planning can be formulated as a receding-horizon optimization problem. Given a state s_0, planning horizon H, and discount factor γ, the planner first generates a set of K candidate actions $\{\mathbf{a}_0^k\}_{k=1}^K$, where each action is sampled from a stochastic single-step action proposer $u(a \mid s)$.

The expected return of each candidate action is computed as $G^k = \mathcal{G}\big(s_0, a_0^k; \hat{T}, \hat{r}, \hat{Q}, H, \gamma\big)$, where $\mathcal{G}$ defines the rollout strategy and return computation under the learned dynamics model $\hat{T}$, combining predicted rewards with terminal value estimates along the simulated trajectory.

Finally, instead of strictly restricting the choice to the single highest-scoring candidate, the executed action a^* is determined by an action selection rule $\mathcal{A}$:

$$a^* = \mathcal{A}\left(\{\mathbf{a}_0^k, G^k\}_{k=1}^K\right). \tag{1}$$

We summarize the overall planning procedure in Algorithm 1, where the specific instantiations of the proposer u, return estimator $\mathcal{G}$, and selection rule $\mathcal{A}$ determine the resulting planning algorithm.

Algorithm 1 General offline model-based planning

Input: Action proposer $u(a \mid s)$, dynamic model $\hat{T}$, value function $\hat{Q}$, reward model $\hat{r}$, return function $\mathcal{G}$, action selection function $\mathcal{A}$, candidate number K, horizon H
Observe initial state s_0
while episode not terminated **do**
  for $k = 1$ **to** K **do**
    $a_0^k \sim u(\cdot \mid \hat{s}_0^k)$ ▷ Action candidate generation
    $G^k = \mathcal{G}\big(s_0, a_0^k; \hat{T}, \hat{r}, \hat{Q}, H, \gamma\big)$, ▷ Rollout the trajectory and compute the return:
  end for
  Select action $a^* = \mathcal{A}\left(\{\mathbf{a}_0^k, G^k\}_{k=1}^K\right)$ ▷ Select an action to be executed
  Execute a^*, observe real next state $s_0 \leftarrow s'$
end while

2.3 Plug-and-play planning methods

To leverage the knowledge encoded in offline RL policies π and value functions, plug-and-play planning methods [13, 16, 17] incorporate action proposers u and value functions $\hat{Q}$ learned via offline RL into the planning process. Model-Based Offline Planning (MBOP) [13] is among the first works to introduce model-based planning to offline scenarios and is among the simplest planning methods. It rolls out K candidate actions using the dynamic models and evaluates their expected return at horizon H. The best action is chosen in a softmax-like manner. We consider the plug-and-play version in this work, where the action proposer and value function are from offline RL algorithms. Learning off-policy with online planning (LOOP) [16] constrains planned action sequences with a KL-divergence regularizer to limit deviation from the action proposer. We consider its offline variant, LOOP-offline, as a plug-and-play offline model-based planning method. Unlike MBOP, LOOP performs ensemble rollouts over future states to capture model uncertainty via the standard deviation of predicted returns. More details on MBOP and LOOP are provided in Appendix C.1.

3 Making planning horizon insensitive

Existing plug-and-play planning algorithms have been shown to improve upon the base policies [16, 17], while appealing, we find that their performance is highly sensitive to the hyperparameter:

planning horizon H. We hypothesize that this sensitivity arises because prior methods employ a **fixed** planning horizon across all states during online evaluation. To investigate this issue, we first present an empirical study in Section 3.1 that illustrates why a fixed horizon can be problematic. We then introduce **Soft Horizon AggRegation for Planning** in Section 3.2, a novel planning algorithm designed to mitigate horizon sensitivity. Finally, in Section 3.3, we demonstrate that our method is substantially more robust to the choice of planning horizon H, leading to improved performance in settings where hyperparameter tuning is impractical.

3.1 Why fixed horizons fail?

The planning horizon determines how far into the future the planner should roll out trajectories, balancing long-term information gain with the accumulation of model errors. However, this trade-off may be state-dependent due to uneven state coverage in the offline dataset.

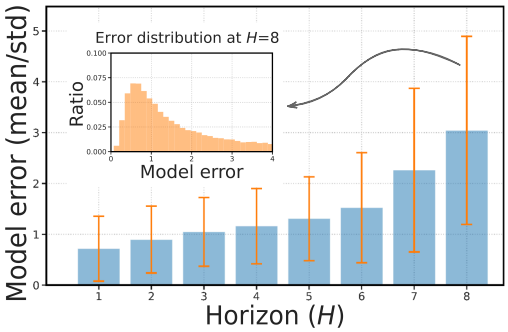


Figure 1: **Optimal planning horizon is state-dependent.** To study why a single fixed horizon can be suboptimal, we analyze model error across different planning horizons on *HalfCheetah-full-replay*. As the horizon increases, the mean error grows, but more importantly, the error variance also increases substantially. This indicates that long-horizon planning is not uniformly harmful across all states.

To investigate this phenomenon, we conduct an experiment comparing the learned dynamics model with the ground-truth dynamics of the environment. On the D4RL [18] *HalfCheetah-v2* environment, we train the dynamics model and the IQL policy using the full-replay dataset. Training details are provided in Appendix D and E. Starting from the same states and using identical action sequences sampled from the policy, we roll out both the learned dynamics model and the real environment, and measure the resulting L2 discrepancies between observations over horizons. As shown in Figure 1, the discrepancy grows as the planning horizon increases, while also varying substantially across states. These results suggest that the optimal planning horizon is state-dependent: some states benefit from longer rollouts that capture more long-term information, whereas others require shorter horizons to mitigate the accumulation of model error. Therefore, previous planning methods that use a fixed horizon across all states require careful online hyperparameter tuning, which limits their practical use.

3.2 SHARP: Soft Horizon AggRegation for Planning

To address the state-dependent model error issue, we propose **Soft Horizon AggRegation for Planning** (SHARP), a plug-and-play, offline model-based planning method that is robust to planning horizon H. Instead of relying on a single, rigid rollout length, the key idea is to dynamically adjust the effective planning horizon for each state by normalizing predicted returns based on ensemble variance. SHARP achieves this through the following steps: (1) simulating diverse future trajectories via ensemble rollouts, (2) estimating horizon-specific returns and precision, (3) soft aggregating these horizon-wise returns using precision as weights, and (4) selecting the final action for execution.

Simulating diverse future trajectories. Similar to LOOP [16], we employ an ensemble of dynamics models. For each model, we maintain P particles, each corresponding to an independent rollout branch. For a candidate action sequence k, future states and actions are sampled according to

$$\hat{s}^k_{h+1,e,p} \sim \hat{T}_e(\cdot \mid \hat{s}^k_{h,e,p}, a^k_{h,e,p}), \quad a^k_{h+1,e,p} \sim u(\cdot \mid \hat{s}^k_{h+1,e,p}), \tag{2}$$

where the rollout is initialized with $\hat{s}^k_{0,e,p} \leftarrow s_0$ and $a^k_{0,e,p} \leftarrow a^k_0$ for all ensemble members e and particles p.

Horizon-specific return and confidence estimation. To safely estimate performance, we calculated the expected return $G^k_{e,p,h}$ at every horizon h for each candidate k, across every branch (ensemble e and particle p):

$$G^k_{e,p,h} = \sum_{t=0}^{h} \gamma^t \hat{r}(\hat{s}^k_{t,e,p}, a^k_{t,e,p}) + \gamma^{h+1} \hat{Q}(\hat{s}^k_{h+1,e,p}, a^k_{h+1,e,p}), \tag{3}$$

Then we calculate the mean μ_h^k and precision ρ_h^k across all ensemble members and particles:

$$\mu_h^k = \text{mean}_{e,p}(G_{e,p,h}^k), \quad \rho_h^k = 1/\text{var}_{e,p}(G_{e,p,h}^k) \tag{4}$$

Soft horizon aggregation. We compute a final, normalized return G^k for each candidate action by taking a precision-weighted sum of the returns across all horizons:

$$G^k = \sum_{h=0}^{H-1} \left(\frac{\rho_h^k}{\sum_{h'=0}^{H-1} \rho_{h'}^k} \right) \mu_h^k \tag{5}$$

Intuitively, a large variance indicates high uncertainty about a specific rollout depth, so that the horizon is downweighted to prevent error accumulation. Conversely, a smaller variance indicates greater certainty, leading to a larger weight being assigned to that horizon's mean return. We call this **soft horizon aggregation** because it "softly" combines return estimates across multiple horizons.

Selecting the final action. Finally, the action is then selected as:

$$a^* = \frac{\sum_{k=1}^K \exp(\kappa G^k) a_0^k}{\sum_{k=1}^K \exp(\kappa G^k)} \tag{6}$$

where κ is a hyperparameter controlling how strongly to emphasize actions with larger estimated return G^k. The full algorithm is presented in Algorithm 2.

Algorithm 2 Soft Horizon AggRegation for Planning

Input: Action proposer $u(a \mid s)$, ensemble of dynamic models $\{\hat{T}_e\}_{e=1}^E$, value function $\hat{Q}$, reward model $\hat{r}$, candidate number K, horizon H, Particle numer P, discount factor γ
Observe initial state s_0
while episode not terminated **do**
  for $k = 1$ **to** K **do**
    $a_0^k \leftarrow u(\cdot \mid s_0)$
    Set $\hat{s}_{0,e,p}^k \leftarrow s_0$, $a_{0,e,p}^k \leftarrow a_0^k \quad \forall\, e, p$
    for $e = 1$ **to** E **do** ▷ Simulate diverse future trajectories
      for $p = 1$ **to** P **do**
        for $h = 0$ **to** $H - 1$ **do**
          Sample next state $\hat{s}_{h+1,e,p}^k \sim \hat{T}_e(\cdot \mid \hat{s}_{h,e,p}^k, a_{h,e,p}^k)$
          Sample next action $a_{h+1,e,p}^k \sim u(\cdot \mid \hat{s}_{h+1,e,p}^k)$
          $G_{e,p,h}^k = \sum_{t=0}^h \gamma^t \hat{r}(\hat{s}_{t,e,p}^k, a_{t,e,p}^k) + \gamma^{h+1} \hat{Q}(\hat{s}_{h+1,e,p}^k, a_{h+1,e,p}^k)$
        end for
      end for
      $\mu_h^k = \text{mean}_{e,p}(G_{e,p,h}^k)$ ▷ Horizon-Specific Return
      $\rho_h^k = 1/\text{var}_{e,p}(G_{e,p,h}^k)$ ▷ Horizon-Specific Precision
    end for
    $G^k = \sum_{h=0}^{H-1} \left(\frac{\rho_{k,h}}{\sum_{h'=0}^{H-1} \rho_{k,h'}} \right) \mu_h^k$ ▷ Soft Horizon Aggregation
  end for
  Select $a^* = \frac{\sum_{k=1}^K \exp(\kappa G^k) a_0^k}{\sum_{k=1}^K \exp(\kappa G^k)}$ ▷ Weighted action selection
  Execute a^*, observe real next state $s_0 \leftarrow s'$
end while

3.3 Experiment

We compare our proposed SHARP with prior plug-and-play planning methods, MBOP [13] and LOOP [16]. We train Q-functions using two classic offline RL algorithms, IQL and CQL, and use their trained policies as the action proposer q. We evaluate on the D4RL locomotion benchmarks [18] across three environments (*HalfCheetah*, *Hopper*, and *Walker2d*), each with five dataset configurations (D): random (R), medium (M), medium-replay (MR), medium-expert (ME), and full-replay (FR).

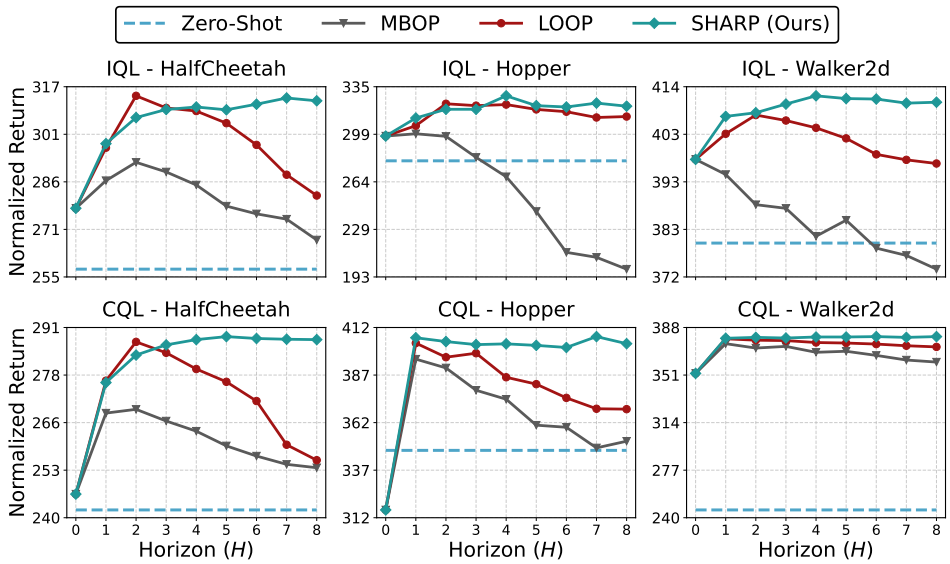


Figure 2: **SHARP is robust to different horizons.** We vary the test-time planning horizon H for IQL- and CQL-based planners on D4RL locomotion tasks. MBOP and LOOP are sensitive to horizon choice, often improving at short horizons but degrading as H increases due to accumulated model errors. In contrast, SHARP maintains stable performance across horizons.

Details of the environments, datasets, and offline RL training setup, including the IQL and CQL training details, are provided in Appendix C.3 and Appendix D.

To examine the stability of all methods across different planning horizons, we vary the planning horizon H from 1 to 8 and evaluate their effects on planning performance. For each horizon, we report the best performance obtained by searching over $\kappa \in \{0.1, 0.5, 1.0, 5.0, 10.0\}$ averaged over 3 seeds and 20 evaluation episodes.

The results in Figure 2 show that the planning horizon H plays a critical yet double-edged role in the performance of existing plug-and-play methods such as MBOP and LOOP. As the planning horizon increases, performance initially improves, highlighting the benefits of longer-horizon planning. However, beyond a certain point, the performance of MBOP and LOOP declines significantly. This degradation arises because the compounding of model errors eventually outweighs the advantages of deeper planning. As a result, these methods are highly sensitive to the choice of H, requiring careful environment-specific hyperparameter tuning.

In contrast, our proposed SHARP addresses this limitation through soft-horizon aggregation. This mechanism enables the use of a universally large maximum horizon H by automatically adapting the contribution of deeper planning steps. Specifically, rollouts are weighted according to uncertainty estimates derived from the ensemble branches, allowing the method to progressively down-weight distant and error-prone predictions. As a result, SHARP maintains stable and strong performance even as the planning horizon increases, effectively eliminating the need for manual horizon tuning.

To rigorously evaluate planning performance without online hyperparameter tuning, we conduct a grid search over the planning hyperparameters, varying the horizon H from 1 to 8 and $\kappa \in \{0.1, 0.5, 1.0, 5.0, 10.0\}$ and report the average performance across all configurations in Table 1. Under this challenging evaluation protocol, our method consistently outperforms both the original offline RL policies (Original) and the MBOP and LOOP baselines. The consistent gains across both IQL- and CQL-based policies demonstrate that our approach not only improves performance substantially but also significantly reduces sensitivity to online hyperparameter tuning. We report the best performance after tuning for all planning methods in Appendix B.1.

4 Does a stronger policy make a better action proposer?

The action proposer is a fundamental yet underexplored component in offline model-based planning. Existing works [16, 17] typically adopt policies trained via offline RL as action proposers, motivated

Table 1: **SHARP achieves the best test-time planning performance without tuning H or κ.** We report the normalized return averaged over all tested planning horizons H and action-selection temperatures κ, simulating a setting where online selection of the best planning configuration is unavailable. SHARP achieves stronger average performance than both the zero-shot offline RL policies and plug-and-play planning baselines across IQL and CQL.

Environment	D	IQL				CQL			
		Zero-shot	MBOP	LOOP	SHARP (Ours)	Zero-shot	MBOP	LOOP	SHARP (Ours)
HalfCheetah	R	6.8	17.7	19.0	**19.3**	4.4	4.2	5.2	**5.4**
	M	46.9	51.6	53.6	**54.5**	44.9	50.3	52.4	**53.4**
	MR	41.7	43.1	46.4	**47.9**	29.3	31.0	32.3	**33.0**
	ME	90.7	72.5	**92.7**	77.4	91.5	92.3	99.3	**102.3**
	FR	71.7	71.8	74.8	**80.6**	72.4	76.5	78.2	**81.0**
Hopper	R	7.6	7.8	**7.9**	7.8	4.3	**5.1**	5.0	4.5
	M	51.8	41.2	56.4	**58.9**	58.5	63.1	69.1	**71.9**
	MR	55.9	70.7	87.9	**88.2**	20.0	79.2	84.2	**93.3**
	ME	**70.9**	30.9	34.7	31.5	62.6	105.9	**107.9**	106.0
	FR	**93.4**	69.1	91.6	91.4	101.0	103.5	104.3	**104.6**
Walker2d	R	4.8	4.8	5.1	**5.5**	**0.1**	**0.1**	**0.1**	**0.1**
	M	83.5	80.7	87.6	**89.3**	73.5	81.7	83.5	**84.0**
	MR	83.9	82.5	92.8	**95.1**	74.2	79.7	84.4	**86.2**
	ME	**109.5**	108.2	109.1	108.6	108.6	109.2	**110.0**	**110.0**
	FR	97.9	96.5	99.9	**100.7**	91.1	93.5	94.2	**95.2**

by the intuition that higher-quality policies should naturally yield better planning performance. Under this view, one would expect that improving the policy, through better objectives or hyperparameter tuning, should directly translate into improved planning outcomes.

In this section, we re-examine this assumption by systematically varying both the policy extraction method and the degree of optimization, and evaluating the resulting policies as action proposers during planning. Specifically, we consider two policy-extraction objectives (described in Section 4.1) and analyze how planning performance varies with optimization strength (Section 4.2).

4.1 Policy extraction in Offline-RL

Policy extraction is a crucial design choice in Offline-RL, aiming to maximize policy value while staying close to the behavior policy represented in the dataset. We consider the following two widely used objectives for offline policy extraction:

(1) Weighted Behavioral Cloning (AWR) [19]. We adopt advantage-weighted regression (AWR), a simple and widely used offline policy extraction method:

$$\max_{\pi} \; \mathcal{J}_{\text{AWR}}(\pi) = \mathbb{E}_{s,a\sim\mathcal{D}}\left[e^{\alpha(Q(s,a)-V(s))}\log\pi(a \mid s)\right], \tag{7}$$

where α is a temperature parameter. As $\alpha \to 0$, the objective reduces to standard behavioral cloning, while larger α emphasizes high-advantage actions.

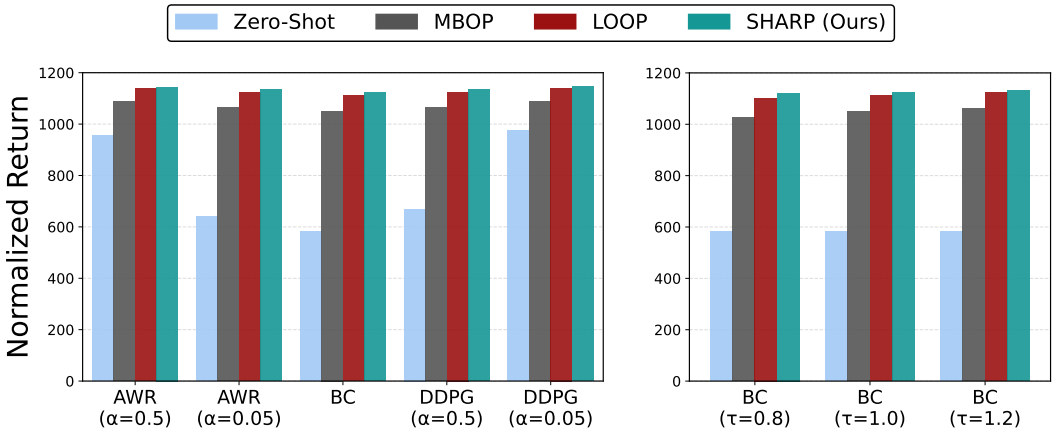
(2) Behavior-Constrained Policy Gradient (DDPG+BC) [9]. We also consider a behavior-constrained objective that combines policy gradient with behavior cloning:

$$\max_{\pi} \; \mathcal{J}_{\text{DDPG+BC}}(\pi) = \mathbb{E}_{s,a\sim\mathcal{D}}\left[Q(s,\mu^{\pi}(s)) + \alpha\log\pi(a \mid s)\right], \tag{8}$$

where $\mu^{\pi}(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[a]$. When $\alpha \to \infty$, the objective reduces to behavioral cloning, where a smaller α places more emphasis on maximizing Q.

4.2 Proposal policy experiment

To investigate whether the quality of the action proposer truly influences planning performance, we apply both AWR and DDPG+BC with varying values of α, using value functions trained by IQL to extract the policy π. We then use the resulting policy as the action proposer for all planning methods, MBOP, LOOP, SHARP (Ours). For each planning method, following prior works [16, 17], we tune the planning hyperparameter over $H \in \{2, 4\}$ and $\kappa \in \{0.1, 0.5, 1.0, 5.0, 10.0\}$, and report the best-performing setting averaged over 3 seeds and 20 evaluation episodes.



(a) Action Proposer Analysis

(b) BC Temperature (τ) Sensitivity

Figure 3: **Planning is robust to the choice of action proposer.** We compare action proposers obtained from AWR, DDPG+BC, and BC using IQL-trained value functions. Despite large differences in the zero-shot performance of the RL policies, planning performance remains similar across different action proposers and BC temperatures.

The results in Figure 3a show that different α leads to substantial variation in zero-shot policy performance, as expected. Surprisingly, however, the planning performance remains largely unchanged across different action proposers and values of α, despite the wide gap in their zero-shot performance. Moreover, we find that even a simple behavior cloning (BC) policy achieves planning performance comparable to that of more sophisticated policy extraction methods. These results suggest that, although policy extraction methods require careful objective and hyperparameter tuning to achieve strong zero-shot performance, such optimization plays a much less important role when the policy is used as an action proposer for planning. We include the analyses across dataset types in Appendix B.3.

We further analyze the effect of varying the BC temperature on planning performance, where higher temperatures correspond to larger action distributions. The results in Figure 3b show that planning performance is likewise largely insensitive to the BC temperature. Taken together, these findings reveal a surprising decoupling between policy quality and planning performance. In practice, the role of the action proposer appears to be primarily to provide a sufficiently diverse set of candidate actions, while the policy's optimality becomes largely irrelevant once sufficient coverage is achieved.

Together, these findings substantially simplify action proposer design: simple BC is sufficient—sophisticated policy extraction methods and additional hyperparameter tuning are not needed.

5 Comparsion with end-to-end planning methods

We combine the findings from the previous sections to derive SHARP-BC, which combines soft horizon aggregation with a simple BC policy as an action proposer. We compare SHARP-BC against classical end-to-end planning baselines, *e.g.*, sequence-based methods (Diffuser [20], DT [21]) and value-based approaches (MBOP [13], IQL-TD-MPC [22]). See Appendix C.2 for more details.

We report the performance of SHARP-BC with and without hyperparameter search (by averaging over all H from 1 to 8 and $\kappa \in \{0.1, 0.5, 1.0, 5.0, 10.0\}$). The results are presented in Table 2, where * denotes the default setting without hyperparameter tuning and unmarked SHARP-BC columns denote tuned settings. Importantly, the baseline methods report results using their best-performing hyperparameters obtained through online hyperparameter search. In contrast, even without any tuning, our method consistently achieves competitive or superior performance relative to these fully tuned baselines. Moreover, when a tuning budget is available, SHARP-BC can further leverage hyperparameter search to attain even stronger performance.

Overall, these findings demonstrate that our proposed SHARP-BC requires less tuning and achieves better planning, and can scale effectively with additional tuning resources.

Table 2: **Performance comparison with end-to-end planning baselines on D4RL locomotion tasks.** We compare SHARP-BC against sequence-based and value-based offline planning baselines. The baseline results are reported using their best-performing settings. For SHARP-BC, columns marked with * denote the default setting without hyperparameter tuning, while unmarked columns use a full tuning budget. Results show that SHARP-BC remains competitive without tuning and further improves when tuning is available.

Environment	Dataset	Sequence-based		Value-based		SHARP-BC (Ours)			
		DT	Diffuser	IQL-TD-MPC	MBOP	IQL*	IQL	CQL*	CQL
HalfCheetah	M	42.6	42.8	57.4	44.6	53.9	**59.6**	52.9	57.2
	MR	36.6	37.7	49.2	42.3	48.5	**52.0**	31.7	32.8
	ME	86.8	88.9	44.8	105.9	95.8	103.4	102.8	**106.9**
Hopper	M	67.6	74.3	–	12.4	68.2	83.1	73.6	**88.4**
	MR	82.7	93.6	–	48.8	92.2	99.2	96.0	**100.1**
	ME	107.6	103.3	–	55.1	101.4	109.7	106.7	**110.7**
Walker2d	M	74.0	79.6	–	9.7	85.4	**91.0**	83.7	84.8
	MR	66.6	70.6	–	41.0	87.6	**95.2**	85.7	89.1
	ME	108.1	106.9	–	70.2	109.8	**110.4**	109.9	110.2
Average		74.7	77.5	50.5	47.8	82.5	**89.3**	82.5	86.7

6 Related work

Offline RL [1] learns policies from static datasets without online interaction. Its core challenge is distribution shift: the learned policy may choose out-of-distribution (OOD) actions with unreliable value estimates. Prior work addresses this through conservative value estimation [8, 23], behavior regularization [24, 9, 10], in-sample learning [7, 25, 26], and uncertainty-aware regularization [27, 28]. Several studies further show that offline RL performance depends strongly on implementation and policy extraction design choices [29, 10, 30]. In contrast, we focus on test-time planning.

Offline model-based planning learns a dynamics model from static data and performs test-time planning over candidate actions. Unlike standard offline RL, these methods adapt decisions online through search. MBOP [13] is an early example, while later approaches improve robustness to model error through uncertainty-aware planning. MOPP [14] removes uncertain branches, IQL-TD-MPC [22] combines TD-MPC with offline RL objectives, and plug-and-play planners such as LOOP [16] and RefPlan [17] integrate offline RL policies and value functions into planning.

Offline model-based RL uses learned dynamics models to improve policy optimization during training rather than for test-time planning. Since policies can exploit model errors on OOD state-action pairs, prior methods employ pessimistic rollouts [31–33], conservative value regularization [34], and adversarial dynamics models [35]. Most related to our work, STEVE [36] adapts rollout horizons using ensemble uncertainty estimates; we instead apply this principle to test-time planning.

7 Conclusions

We introduce Soft Horizon AggRegation for Planning (SHARP), which addresses horizon sensitivity by dynamically weighting returns based on model uncertainty, eliminating the need for manually tuned, state-invariant horizons. We further show that sophisticated policy extraction is often unnecessary, as a simple behavior cloning (BC) proposer can achieve comparable planning performance. Combining these insights, we propose SHARP-BC, a robust plug-and-play offline planning method that consistently outperforms existing baselines across diverse tasks and dataset settings. Overall, our approach simplifies offline planning by reducing reliance on extensive hyperparameter tuning while maintaining strong performance.

References

[1] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[2] Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, et al. $\pi^*_{0.6}$: a vla that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025.

[3] Yevgen Chebotar, Quan Vuong, Karol Hausman, Fei Xia, Yao Lu, Alex Irpan, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, et al. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions. In *Conference on Robot Learning*, 2023.

[4] Tianyu Shi, Dong Chen, Kaian Chen, and Zhaojian Li. Offline reinforcement learning for autonomous driving with safety and exploration enhancement. *arXiv preprint arXiv:2110.07067*, 2021.

[5] Antonio Guillen-Perez. From imitation to optimization: A comparative study of offline learning for autonomous driving. *arXiv preprint arXiv:2508.07029*, 2025.

[6] Shengpu Tang and Jenna Wiens. Model selection for offline reinforcement learning: Practical considerations for healthcare settings. In *Machine Learning for Healthcare Conference*, 2021.

[7] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022.

[8] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in neural information processing systems*, 2020.

[9] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *Advances in neural information processing systems*, 2021.

[10] Denis Tarasov, Vladislav Kurenkov, Alexander Nikulin, and Sergey Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

[11] Dibya Ghosh, Anurag Ajay, Pulkit Agrawal, and Sergey Levine. Offline rl policies should be trained to be adaptive. In *International Conference on Machine Learning*, 2022.

[12] Xiong-Hui Chen, Yang Yu, Qingyang Li, Fan-Ming Luo, Zhiwei Qin, Wenjie Shang, and Jieping Ye. Offline model-based adaptable policy learning. In *Advances in Neural Information Processing Systems*, 2021.

[13] Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning. In *International Conference on Learning Representations*, 2021.

[14] Xianyuan Zhan, Xiangyu Zhu, and Haoran Xu. Model-based offline planning with trajectory pruning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, 2022.

[15] Jacques Richalet, André Rault, Jean-Louis Testud, and Jean Papon. Model predictive heuristic control. *Automatica (journal of IFAC)*, 14, 1978.

[16] Harshit Sikchi, Wenxuan Zhou, and David Held. Learning off-policy with online planning. In *5th Annual Conference on Robot Learning*, 2021.

[17] Jihwan Jeong, Xiaoyu Wang, Jingmin Wang, Scott Sanner, and Pascal Poupart. Reflect-then-plan: Offline model-based planning through a doubly bayesian lens. In *Forty-second International Conference on Machine Learning*, 2025.

[18] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[19] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[20] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022.

[21] 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 *Advances in Neural Information Processing Systems*, 2021.

[22] Rohan Chitnis, Yingchen Xu, Bobak Hashemi, Lucas Lehnert, Urun Dogan, Zheqing Zhu, and Olivier Delalleau. Iql-td-mpc: Implicit q-learning for hierarchical model predictive control. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[23] Ching-An Cheng, Tengyang Xie, Nan Jiang, and Alekh Agarwal. Adversarially trained actor critic for offline reinforcement learning. In *International Conference on Machine Learning*, 2022.

[24] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[25] Haoran Xu, Li Jiang, Jianxiong Li, Zhuoran Yang, Zhaoran Wang, Victor Wai Kin Chan, and Xianyuan Zhan. Offline rl with no ood actions: In-sample learning via implicit value regularization. In *International Conference on Learning Representations*, 2023.

[26] Divyansh Garg, Joey Hejna, Matthieu Geist, and Stefano Ermon. Extreme q-learning: Maxent RL without entropy. In *The Eleventh International Conference on Learning Representations*, 2023.

[27] Gaon An, Seungyong Moon, Jang-Hyun Kim, and Hyun Oh Song. Uncertainty-based offline reinforcement learning with diversified q-ensemble. In *Advances in neural information processing systems*, 2021.

[28] Seyed Kamyar Seyed Ghasemipour, Shixiang Shane Gu, and Ofir Nachum. Why so pessimistic? estimating uncertainties for offline RL through ensembles, and why their independence matters. In *Advances in Neural Information Processing Systems*, 2022.

[29] Yuwei Fu, Di Wu, and Benoit Boulet. A closer look at offline RL agents. In *Advances in Neural Information Processing Systems*, 2022.

[30] Seohong Park, Kevin Frans, Sergey Levine, and Aviral Kumar. Is value learning really the main bottleneck in offline RL? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[31] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. In *Advances in neural information processing systems*, 2020.

[32] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. In *Advances in neural information processing systems*, 2020.

[33] Cong Lu, Philip Ball, Jack Parker-Holder, Michael Osborne, and Stephen J. Roberts. Revisiting design choices in offline model based reinforcement learning. In *International Conference on Learning Representations*, 2022.

[34] Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. In *Advances in neural information processing systems*, 2021.

[35] Marc Rigter, Bruno Lacerda, and Nick Hawes. Rambo-rl: Robust adversarial model-based offline reinforcement learning. In *Advances in neural information processing systems*, 2022.

[36] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, 2018.

[37] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[38] Soichiro Nishimori. Jax-corl: Clean single-file implementations of offline rl algorithms in jax. 2024. URL `https://github.com/nissymori/JAX-CORL`.

Appendix

Table of Contents

A Limitations

The primary goal of this work is to design a tuning-free, plug-and-play offline reinforcement learning algorithm. While our proposed method, SHARP-BC, effectively eliminates the need for online tuning of the planning horizon and simplifies the selection of an action proposer, certain limitations remain that provide avenues for future investigation:

Sensitivity to action-selection temperature (κ): Although we have successfully addressed the challenges of horizon selection and action proposer design, our current method remains slightly sensitive to the hyperparameter κ. This sensitivity led to lower performance in some experimental results when evaluated under a strictly tuning-free protocol. Developing a mechanism to automatically adapt or eliminate the need for tuning κ is a critical area for future work.

Hyperparameters in value function learning: To move a significant step closer toward a truly end-to-end tuning-free model-based planning framework, it will be necessary to investigate the hyperparameters involved in training the underlying value functions (e.g., Q-function learning). Currently, these still rely on standard offline RL training configurations, which may not be optimal for all planning scenarios. By addressing these remaining sensitivities, we aim to further simplify the deployment of offline model-based agents in practical, real-world applications where online evaluation is impossible.

Focus on non-generative policies: To simplify the scope of this work, we focus exclusively on non-generative policies in this paper. Nevertheless, generative models are capable of producing diverse action distributions and may therefore serve as effective action proposers. We leave the exploration of generative models in this context to future work.

B Additional experimental results

B.1 Best performance after online hyperparameter tuning

In the main paper, Table 1 evaluates robustness by averaging performance over all tested planning horizons H and action-selection temperatures κ. This reflects a setting where online hyperparameter selection is unavailable. Here, we also report the max performance after hyper search.

As shown in Table 3, SHARP still performs better than baselines by doing soft horizon aggregation.

Table 3: **Max planning performance.** We report the max normalized return across all evaluated settings for each planning method.

Environment	Dataset	IQL				CQL			
		Zero-shot	MBOP	LOOP	SHARP (Ours)	Zero-shot	MBOP	LOOP	SHARP (Ours)
HalfCheetah	R	6.8	21.4±0.6	23.2±0.8	**23.8±1.0**	4.4	5.7±1.4	6.7±1.2	**7.2±1.5**
	M	46.9	55.3±0.2	58.4±0.4	**60.1±0.3**	44.9	53.1±0.3	55.8±0.4	**57.8±0.5**
	MR	41.7	45.5±0.1	48.9±0.2	**52.5±0.6**	29.3	32.4±22.9	33.5±23.6	**34.2±24.3**
	ME	90.7	91.1±1.7	**101.5±1.0**	93.4±0.4	91.5	99.3±1.0	**107.7±0.3**	107.0±0.3
	FR	71.7	79.3±1.3	83.6±1.1	**83.7±1.2**	72.4	79.6±0.5	**83.4±0.6**	82.5±0.2
Hopper	R	7.6	7.9±0.2	**8.2±0.4**	**8.2±0.3**	4.3	**6.1±0.7**	5.8±0.4	5.0±0.6
	M	51.8	53.4±3.3	73.8±3.7	**80.4±6.7**	58.5	74.4±3.7	82.7±3.6	**84.9±2.3**
	MR	55.9	98.7±0.7	**99.1±0.2**	98.8±0.3	20.0	99.5±0.6	100.1±0.7	**100.2±0.6**
	ME	**70.9**	39.3±3.1	41.7±8.9	36.7±2.4	62.6	111.3±0.1	110.9±0.3	**111.4±0.3**
	FR	93.4	102.7±1.2	**106.0±0.1**	104.8±0.3	101.0	105.4±0.7	**106.2±0.2**	**106.2±0.5**
Walker2d	R	4.8	5.7±0.7	**6.5±0.6**	**6.5±0.6**	**0.1**	**0.1±0.0**	**0.1±0.0**	**0.1±0.0**
	M	83.5	87.0±2.0	92.4±0.3	**93.7±0.5**	73.5	84.2±0.3	84.6±0.2	**84.7±0.1**
	MR	83.9	93.1±1.3	97.2±1.1	**98.7±1.1**	74.2	86.4±0.4	87.8±1.7	**89.3±0.6**
	ME	109.5	108.9±0.1	**110.1±0.4**	109.4±0.2	108.6	110.0±0.0	**110.4±0.3**	**110.4±0.3**
	FR	97.9	99.9±1.8	103.5±0.3	**104.2±0.2**	91.1	95.1±0.9	96.7±0.7	**97.1±1.6**

Table 4: **Mean planning performance.** We report the mean normalized return across all evaluated settings for each planning method.

Environment	Dataset	IQL				CQL			
		Zero-shot	MBOP	LOOP	SHARP (Ours)	Zero-shot	MBOP	LOOP	SHARP (Ours)
HalfCheetah	R	6.8	17.7±0.7	19.0±0.9	**19.3±0.9**	4.4	4.2±1.1	5.2±1.2	**5.4±1.1**
	M	46.9	51.6±0.2	53.6±0.1	**54.5±0.2**	44.9	50.3±0.2	52.4±0.3	**53.4±0.3**
	MR	41.7	43.1±0.5	46.4±0.2	**47.9±0.2**	29.3	31.0±21.9	32.3±22.8	**33.0±23.4**
	ME	90.7	72.5±3.2	**92.7±1.2**	77.4±6.0	91.5	92.3±2.9	99.3±1.3	**102.3±1.3**
	FR	71.7	71.8±1.3	74.8±1.0	**80.6±1.1**	72.4	76.5±1.2	78.2±1.5	**81.0±0.7**
Hopper	R	7.6	7.8±0.2	**7.9±0.3**	7.8±0.3	4.3	**5.1±0.6**	5.0±0.6	4.5±0.9
	M	51.8	41.2±2.3	56.4±4.0	**58.9±6.2**	58.5	63.1±2.8	69.1±3.4	**71.9±3.1**
	MR	55.9	70.7±3.0	87.9±1.2	**88.2±1.0**	20.0	79.2±6.8	84.2±6.4	**93.3±2.5**
	ME	70.9	30.9±2.8	**34.7±3.5**	31.5±3.6	62.6	105.9±4.0	**107.9±2.8**	106.0±4.4
	FR	**93.4**	69.1±6.1	91.6±2.9	91.4±3.5	101.0	103.5±0.3	104.3±0.5	**104.6±0.5**
Walker2d	R	4.8	4.8±0.6	5.1±0.5	**5.5±0.6**	**0.1**	**0.1±0.0**	**0.1±0.0**	**0.1±0.0**
	M	83.5	80.7±3.0	87.6±1.7	**89.3±1.6**	73.5	81.7±1.2	83.5±0.5	**84.0±0.3**
	MR	83.9	82.5±3.5	92.8±2.0	**95.1±1.9**	74.2	79.7±3.3	84.4±2.1	**86.2±1.6**
	ME	109.5	108.2±0.2	**109.1±0.2**	108.6±0.2	108.6	109.2±0.2	**110.0±0.2**	**110.0±0.2**
	FR	97.9	96.5±1.5	99.9±0.6	**100.7±0.5**	91.1	93.5±0.9	94.2±0.9	**95.2±1.1**

B.2 Mean performance across all hyperparameter settings

In the paper, we report the mean planning performance across all hyperparameter settings. Here we report the mean and standard deviation across 3 seeds in Table 4.

B.3 Action proposer analysis across dataset types

In Figure 3, we have shown that using BC as an action proposer yields comparable planning performance. Here, we further compare different action proposers on varying dataset quality. The results in Figure 4 indicate that the correlation between planning performance and zero-shot performance occurs in low-quality datasets (Random). This highlights that when the dataset quality is sufficient, using BC as an action proposer is the simplest option at no cost.

C Baselines and dataset descriptions

C.1 Plug-and-play planning methods

To leverage the knowledge encoded in offline RL policies π and value functions, plug-and-play planning methods [13, 16, 17] incorporate action proposers u and value functions $\hat{Q}$ learned via offline RL into the planning process. Here, we discuss planning algorithms that can be used in a plug-and-play style.

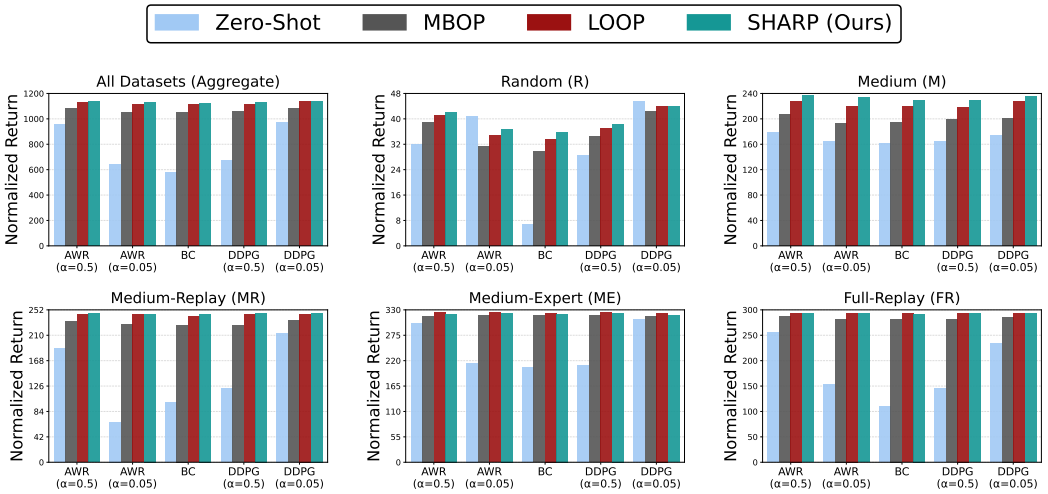


Figure 4: **Planning robustness over action proposers on varying dataset quality.**

MBOP. Model-Based Offline Planning (MBOP) [13], one of the first works to introduce model-based planning to offline scenarios, is among the simplest planning methods. They train an ensemble of dynamic models $\{\hat{T}_e\}_{e=1}^{E}$, a behavior cloning policy as an action proposer u, and use return-to-go as the objective to optimize the value functions. The trajectory is rolled out as:

$$a_{h+1}^k \sim u(\cdot \mid \hat{s}_{h+1}^k), \quad \hat{s}_{h+1}^k \sim \hat{T}_e(\cdot \mid \hat{s}_h^k, a_h^k) \quad e = k \bmod E \tag{9}$$

with $\hat{s}_0^k = s_0, a_0^k = a^k$. The return is estimated as:

$$G^k = \sum_{h=0}^{H-1} \gamma^h \hat{r}(\hat{s}_h^k, a_h^k) + \gamma^H \hat{Q}(\hat{s}_H^k, a_H^k), \tag{10}$$

The action selection rule $\mathcal{A}$ is deined as:

$$\mathcal{A}\left(\{\mathbf{a}_{0:H-1}^k, G^k\}_{k=1}^K\right) = \frac{\sum_{k=1}^K \exp(\kappa G^k) a_0^k}{\sum_{k=1}^K \exp(\kappa G^k)}, \tag{11}$$

where κ is a hyperparameter controlling how strongly to emphasize actions with larger estimated return G^k.

We consider the plug-and-play version in this work.

LOOP. Learning Off-Policy with Online Planning (LOOP) [16] constrains the planned action sequence using a KL-divergence regularization term to prevent the executed actions from deviating excessively from the action proposer. In this work, we consider their proposed offline variant, LOOP-offline, as a plug-and-play method for offline model-based planning. Unlike MBOP, LOOP rolls out an ensemble of future states $\hat{s}_{h,e,p}^k$, where e indexes the dynamics model and p indexes the particle, *i.e.*, different rollout samples generated from the same dynamics model. By performing multiple rollouts across the dynamics ensembles, the method accounts for model uncertainty through the standard deviation of predicted returns. The return function is estimated as:

$$\begin{aligned} G_{p,e}^k &= \sum_{h=0}^{H-1} \gamma^h \hat{r}(\hat{s}_{h,e,p}^k, a_{h,e,p}^k) + \gamma^H \hat{Q}(\hat{s}_{H,e,p}^k, a_{H,e,p}^k), \\ G_e^k &= \text{mean}_p(G_{e,p}^k) - \beta_{pess} \cdot \text{std}_p(G_{e,p}^k), G^k = \text{mean}_e(G_e^k), \end{aligned} \tag{12}$$

where β_{pess} controls the weight of the penalty. The action selection rule $\mathcal{A}$ is the same one as in MBOP.

C.2 End-to-dnd planning baselines

To provide a comprehensive comparison, we select methodologies that represent two paradigms in offline reinforcement learning and planning: sequence-based and value-based approaches. Below, we provide a brief overview of the core mechanics of each baseline.

Sequence-based planning methods reformulate the reinforcement learning and planning problem as a sequence modeling or generative task, typically leveraging generative models to co-learn the planning process and the dynamic model.

Decision Transformer (DT) [21]: Decision Transformer abstracts the offline RL problem into a conditional sequence modeling task. Rather than explicitly fitting value functions or computing temporal difference errors, DT feeds sequences of states, actions, and desired returns (returns-to-go) into a causal transformer architecture. By conditioning on a high target return, the model generates action sequences intended to achieve that outcome, effectively turning planning into autoregressive generation.

Diffuser [20]: Diffuser takes a fundamentally different generative approach by conceptualizing planning as a denoising diffusion process. It models the joint trajectory of states and actions, iteratively refining a sequence of purely Gaussian noise into a physically plausible and high-reward trajectory. Because the diffusion process operates simultaneously over the entire trajectory, it provides a sequence-level perspective on long-horizon planning.

Value-based planning methods rely on learning a separate dynamic model and leveraging it for test-time planning; SHARP falls into this category.

MBOP [13] is discussed in Appendix C.1. We use its original version as the end-to-end baseline.

IQL-TD-MPC [22]: This approach integrates the conservative value estimation of Implicit Q-Learning (IQL) with the latent-space planning capabilities of TD-MPC (Temporal Difference Learning for Model Predictive Control). By leveraging IQL, the method obtains robust value estimates from offline data without querying out-of-distribution actions. These learned representations and value functions are then used to perform efficient, derivative-free trajectory optimization in a learned latent space during inference.

C.3 Tasks and dataset

We evaluate our method on three standard MuJoCo [37] locomotion environments from D4RL [18]: *halfcheetah*, *hopper*, and *walker2d*. These continuous-control tasks are commonly used in offline RL and provide long-horizon decision-making problems for evaluating model-based planning.

For each environment, we use five dataset types: **random** (R), collected by an untrained policy; **medium** (M), collected by a partially trained policy; **medium-replay** (MR), containing the replay buffer accumulated during training to medium performance; **medium-expert** (ME), mixing medium-level and expert demonstrations; and **full-replay** (FR), containing the full replay buffer from scratch to expert performance. These 15 task–dataset combinations span different levels of data quality, behavioral coverage, and state-action distribution shift, allowing us to assess whether planning methods remain robust across diverse offline data regimes.

D Offline RL implementation details

We train both IQL and CQL following Jax-CORL's [38] codebase and hyperparameters. For policy-extraction experiments, all policies (BC, AWR, and DDPG+BC) follow IQL's policy implementation. The hyperparameters are reported in Table 5.

For all zero-shot performance, the policy π will be set to deterministic (τ=0). When the policy π is used as an action proposer u, it will be set to stochastic (τ=1).

E Planning implementation details

We train dynamic models following the setting of previous work [16]. The hyperparameters are provided in Table 6.

For all the plug-and-play planning methods, we set K=32, σ=0.01, and action mixture β=0. For LOOP and SHARP, we set particle P=4. We set β_{pess}=0.1 for LOOP.

Table 5: **Offline RL training hyperparameters.** We list the IQL, CQL, and policy extraction hyperparameters used in all experiments, following the Jax-CORL implementation.

Method	Hyperparameter	Value
IQL	Steps	1M
	Batch size	256
	Target update rate	5e-3
	Actor learning rate	3e-4
	Value learning rate	3e-4
	Critic learning rate	3e-4
	Learning rate decay	Cosine
	Expectile (τ)	0.7 (0.5 for hopper-me, walker2d-mr)
	AWR alpha (β)	3.0 (6.0 for hopper-me)
	Actor hidden dim	[256, 256]
	Critic/Value hidden dim	[256, 256]
CQL	Steps	1M
	Batch size	256
	Target update rate	5e-3
	Actor lr	1e-4
	Critic lr	3e-4
	Target entropy	$-1 \times$ action_dim
	Actor hidden dim	[256, 256, 256]
	Critic hidden dim	[256, 256, 256]
	Lagrange	False
	CQL alpha	10
	CQL num action	10
BC/AWR/DDPG+BC	Steps	1M
	Batch size	256
	Learning rate	3e-4
	Actor hidden dim	[256, 256]
	Policy type	Stochastic

F Experiments compute resources

We performed all experiments on the workstations listed in Table 7.

Table 6: **Hyperparameters for learned dynamics and reward models.** We use the same network architecture and training configuration across all MuJoCo tasks. The dynamics model is trained as a stochastic ensemble, while the reward model is trained as a deterministic predictor.

Model	Hyperparameter	Value
Dynamic Model	Model type	stochastic ensemble 7
	Hidden dimensions	[200, 200, 200, 200]
	Learning rate	1e-3
	Steps	1M
	Batch size	1024
	Validation ratio	0.01
Reward Model	Model type	deterministic 1
	Hidden dimensions	[200, 200, 200, 200]
	Learning rate	1e-3
	Steps	1M
	Batch size	1024
	Validation ratio	0.01

Table 7: **Computational resources used.** We report the hardware used for training offline RL agents, learning dynamics and reward models, and running planning evaluations.

Workstation	CPU	GPU	RAM
Workstation 1	Intel Xeon W-2255	NVIDIA GeForce RTX 3080 Ti	125 GiB
Workstation 2	Intel Xeon W-2255	NVIDIA GeForce RTX 4070 Ti×2	125 GiB
Workstation 3	Intel Xeon Platinum 8480+	NVIDIA H100x2	503 GiB