LESS TUNING, BETTER PLANNING: SIMPLIFYING OFFLINE REINFORCEMENT LEARNING VIA PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Offline reinforcement learning (RL) aims to learn effective policies entirely from precollected data, making it attractive for domains where online interaction is expensive or unsafe. Yet, existing offline RL methods are highly sensitive to hyperparameters and design choices, often requiring online evaluation to identify a good configuration, undermining the offline setting. This work investigates whether plug-and-play model-based planning can improve both policy performance and robustness to offline RL hyperparameters. We focus on two stages of the planning pipeline: test-time planning and offline policy extraction. For test-time planning, we find that performance is highly sensitive to the planning horizon, reflecting a trade-off between incorporating long-term information and accumulating model error. To address this, we propose Soft Horizon AggRegation for Planning (SHARP), which evaluates candidate actions across multiple horizons and softly aggregates their advantages using uncertainty estimates from an ensemble of dynamics models. For policy extraction, we find that Q-function exploitation can improve zero-shot policy performance but provides little additional benefit when the policy serves as an action proposer for planning, whereas behavioral regularization remains essential. Consequently, simple behavior cloning is sufficient as an action proposer. Building on these findings, we propose SHARP-BC, a plug-and-play planning method that requires no planning-specific hyperparameter tuning. Across D4RL locomotion and Adroit dexterous manipulation tasks, SHARP-BC achieves competitive or superior performance to existing offline planning methods while substantially reducing sensitivity to both planning and underlying offline RL hyperparameters.

1 INTRODUCTION

Offline reinforcement learning (RL; Levine et al. 2020) learns a policy from a precollected dataset, aiming to avoid any online interaction during training. This setting is particularly appealing in safety-critical or resource-constrained domains where online exploration is expensive, risky, or impractical, such as robotic manipulation (Intelligence et al., 2025; Chebotar et al., 2023), autonomous driving (Shi et al., 2021; Guillen-Perez, 2025), and healthcare decision-making (Tang and Wiens, 2021). However, current offline RL algorithms have been shown to exhibit high sensitivity to hyperparameter choices (Paine et al., 2020; Kurenkov and Kolesnikov, 2022). In particular, the design of policy improvement or extraction based on the learned value function can substantially affect performance (Park et al., 2024). Identifying a well-performing configuration often requires online evaluation (Jackson et al., 2025), which undermines the core motivation of offline RL.

Offline model-based planning (Argenson and Dulac-Arnold, 2021; Zhan et al., 2022) offers an alternative way to approximate policy improvement by performing model-predictive control (Richalet et al., 1978) at test time. Instead of relying entirely on the policy extracted during training, it learns a dynamics model from the offline dataset and uses it at deployment to simulate diverse future trajectories, evaluate them using the learned value function, and derive an improved action to execute in the environment. Plug-and-play planning methods (Sikchi et al., 2021; Jeong et al., 2025) further incorporate the policies and value functions trained via offline RL algorithms, and have been shown to improve the policy through additional test-time optimization. **This motivates a broader question: beyond improving performance, can we use plug-and-play planning to also reduce the hyperparameter sensitivity of offline RL?**

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.

To answer this question, we examine the two stages in the plug-and-play planning pipeline separately to identify where the performance gains and hyperparameter sensitivity arise. The first is test-time planning, where the planner combines the dynamics model and value function with the policy as an action proposer to derive an improved action at deployment. The second is offline policy extraction, which distills a learned value function into an action proposer for the planner. By studying these two components separately, we investigate how to make plug-and-play planning robust to its own hyperparameter choices and whether it can reduce sensitivity to offline policy extraction.

First, we find that test-time planning can substantially improve offline RL performance, but it introduces a new source of sensitivity through the planning horizon H. Longer horizons allow the planner to incorporate more future information, but they also accumulate more model error, producing a trade-off in which the optimal horizon varies substantially across states and tasks. Therefore, existing methods can improve sharply at short horizons yet degrade when exceeding certain H, making their performance highly sensitive to horizon selection. To address this sensitivity, we propose **Soft Horizon AggRegation for Planning (SHARP)**, which avoids committing to a single effective planning horizon. SHARP evaluates candidate actions across multiple horizons and softly aggregates their advantages according to uncertainty estimated from an ensemble of dynamics models. By automatically down-weighting unreliable rollout depths, SHARP remains stable across a wide range of maximum planning horizons H , enabling test-time planning to improve performance while being robust to its own hyperparameter choices.

Second, we investigate whether policy improvement through test-time planning can reduce sensitivity to policy extraction. Policy extraction typically trades off two objectives: behavioral regularization, which keeps the policy close to the offline data distribution, and Q-function exploitation, which favors actions with higher predicted value. Balancing these two objectives introduces hyperparameters whose optimal values vary across tasks (Park et al., 2024). Existing plug-and-play methods use task-specific tuned policies as the action proposer, which incurs additional online tuning effort (Sikchi et al., 2021; Jeong et al., 2025). We study how these two objectives affect both zero-shot policy and planning performance, and further compare action proposers obtained from different policy-extraction objectives (Peng et al., 2019; Fujimoto and Gu, 2021). We find that Q-function exploitation improves zero-shot policy performance but provides little benefit as the action proposer for planning, while behavioral regularization remains essential. This suggests that test-time planning removes the need for Q-function exploitation during policy extraction, making behavior cloned (**BC**) policies sufficient action proposers and avoiding the need to tune Q-exploitation strength through online evaluation.

Combining these findings, we propose **SHARP-BC**, which uses a horizon-insensitive planning algorithm with a simple behavior-cloned action proposer. We evaluate SHARP-BC on D4RL (Fu et al., 2020) locomotion and Adroit dexterous manipulation tasks, where it achieves competitive or superior performance to existing offline planning methods while not introducing new sources of hyperparameter sensitivity. More importantly, this robustness can even extend to the underlying offline RL methods. Across different hyperparameter configurations, zero-shot policy performance varies considerably, whereas SHARP-BC remains substantially closer to the best attainable performance. Overall, SHARP-BC enables test-time planning to improve policy performance while simultaneously reducing sensitivity to offline RL hyperparameter choices.

2 PRELIMINARIES

We consider Markov decision processes (MDPs) defined by $M = (\mathcal{S}, \mathcal{A}, T, r, d_0, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are 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; Levine et al. 2020), an agent learns from a fixed dataset $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$ collected by an 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 gives the expected return from taking action a in state s and then following 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 planning learns a dynamics model $\hat{T}(s' \mid s, a)$, a reward model $\hat{r}(s, a)$, an action proposer $u(a \mid s)$, and a value function $\hat{Q}(s, a)$ from the offline dataset. Unlike offline model-based RL (MBRL; Yu et al. 2020; Kidambi et al. 2020), which uses simulated rollouts to improve a policy during offline training, model-based planning performs policy improvement at test time by simulating rollouts, evaluating candidate actions, and selecting the most promising action to execute.

Given an initial state s_0, planning horizon H, and discount factor γ, model-based planning typically performs model predictive control (Richalet et al., 1978) using an MPPI-style sampling-based optimization procedure (Williams et al., 2017). At each environment step, it samples K candidate action sequences $\{\mathbf{a}^k\}_{k=1}^K$ using the action proposer u, where $\mathbf{a}^k = (a_0^k, \ldots, a_H^k)$, and evaluates each sequence using the learned dynamics $\hat{T}$, reward $\hat{r}$, and value models $\hat{Q}$. Specifically, the rollout trajectory is generated following $\hat{s}_{t+1}^k \sim \hat{T}(\cdot \mid \hat{s}_t^k, a_t^k)$, starting from $\hat{s}_0^k = s_0$, where its return is estimated as $G^k = \sum_{t=0}^{H-1} \gamma^t \hat{r}(\hat{s}_t^k, a_t^k) + \gamma^H \hat{Q}(\hat{s}_H^k, a_H^k)$. It then softmaxes the actions according to their predicted returns, $a^* = \frac{\sum_{k=1}^K \exp(\kappa G^k) a_0^k}{\sum_{k=1}^K \exp(\kappa G^k)}$, where κ is an inverse-temperature parameter that controls the weighting on high-return trajectories. Finally, the resulting action a^* is executed in the environment, where planning repeats from the newly observed state until the episode terminates.

2.3 PLUG-AND-PLAY PLANNING METHODS

To leverage the knowledge encoded in offline RL policies and value functions, plug-and-play planning methods (Sikchi et al., 2021; Jeong et al., 2025) incorporate the policy as the action proposer u and value function $\hat{Q}$ learned via offline RL algorithms into the planning process. Model-Based Offline Planning (MBOP; Argenson and Dulac-Arnold 2021) is among the first model-based planning works for offline settings. Following MPPI (Williams et al., 2017), it rolls out K candidate actions with the dynamics models, evaluates their returns at horizon H, and selects an action using softmax weighting. We consider its plug-and-play version, where the action proposer and value function are from offline RL algorithms. Learning off-policy with online planning (LOOP; Sikchi et al. 2021) regularizes planned action sequences with KL divergence to limit deviation from the action proposer. We consider its offline variant, LOOP-offline (hereafter LOOP), as a plug-and-play offline planning method. LOOP performs ensemble rollouts over future states and measures model uncertainty via the standard deviation of predicted returns. More details are provided in Appendix E.2.

3 MAKING PLANNING HORIZON INSENSITIVE

In this work, we aim to leverage plug-and-play planning to reduce the hyperparameter sensitivity of offline RL beyond achieving higher returns. While existing plug-and-play methods have improved the performance of policies learned by offline RL algorithms (Sikchi et al., 2021; Jeong et al., 2025), we find that planning itself introduces a new source of sensitivity through the planning horizon H. The planning horizon H determines how far into the future the planner should roll out trajectories, balancing long-term information gain with the accumulation of model errors. This tradeoff can be state-dependent due to uneven state coverage in the offline dataset (Kidambi et al., 2020; Ni et al., 2026). Therefore, prior plug-and-play planning methods can yield substantial performance at shorter horizons but degrade when the horizon exceeds a certain threshold.

To reduce sensitivity to the planning horizon, we introduce **Soft Horizon AggRegation for Planning (SHARP)**, which adaptively aggregates estimates across multiple rollout horizons (Section 3.1). Across a wide range of maximum horizons, SHARP remains substantially more stable and can improve performance without task-specific online tuning (Section 3.2).

3.1 SHARP: SOFT HORIZON AGGREGATION FOR PLANNING

Previous methods use a fixed planning horizon H across all states, which fails to account for the uneven state coverage in the offline dataset, making performance gains sensitive to the choice of planning horizon. To address this, we propose **Soft Horizon AggRegation for Planning** (SHARP), a plug-and-play, offline model-based planning method that is robust to the 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 advantages and precision, (3) soft-aggregating these horizon-wise advantages using precision as weights, and (4) selecting the final action for execution.

Simulating diverse future trajectories. Following LOOP (Sikchi et al., 2021), we use an ensemble of E dynamics models and simulate P independent rollout particles under each model. For a candidate action $a_0^k \sim u(\cdot \mid s_0)$, future states and actions are sampled according to

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

where rollouts are initialized with $\hat{s}^k_{0,e,p} \leftarrow s_0$ and $a^k_{0,e,p} \leftarrow a^k_0$ for models e and particles p.

Horizon-specific advantage and precision estimation. Model value expansion methods (Buckman et al., 2018; Jeong et al., 2023) use model rollouts to improve return estimation. In planning, however, the goal is not to estimate a trajectory's absolute return, but to compare candidate actions and identify which one to execute. Therefore, we consider each candidate's advantage relative to the others.

For each candidate k, we evaluate its expected return $G^k_{h,e,p}$ at every horizon h across all rollout branches (ensemble e and particle p):

$$G^k_{h,e,p} = \sum_{t=0}^{h-1} \gamma^t \hat{r}(\hat{s}^k_{t,e,p}, a^k_{t,e,p}) + \gamma^h \hat{Q}(\hat{s}^k_{h,e,p}, a^k_{h,e,p}). \tag{2}$$

To compare candidates by their relative performance at each horizon, we subtract a shared baseline averaged over all candidates, ensemble members, and particles to derive their advantages $A^k_{h,e,p}$:

$$A^k_{h,e,p} = G^k_{h,e,p} - B_h, \qquad B_h = \text{mean}_{j,e,p}(G^j_{h,e,p}). \tag{3}$$

We then calculate the mean μ^k_h and precision ρ^k_h across all ensemble members and particles:

$$\mu^k_h = \text{mean}_{e,p}(A^k_{h,e,p}), \quad \rho^k_h = 1/\text{var}_{e,p}(A^k_{h,e,p}). \tag{4}$$

Soft horizon aggregation. For each candidate k, we compute the normalized advantage A^k by taking a precision-weighted sum of the advantages across all horizons:

$$A^k = \sum_{h=1}^{H} \left(\frac{\rho^k_h}{\sum_{h'=1}^{H} \rho^k_{h'}} \right) \mu^k_h. \tag{5}$$

Intuitively, a large variance indicates high uncertainty about a specific rollout depth, so 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 advantage. We call this **soft horizon aggregation** because it "softly" combines advantage estimates across multiple horizons.

Selecting the final action. Finally, following MPPI (Williams et al., 2017), the action is selected as:

$$a^* = \frac{\sum_{k=1}^{K} \exp(\kappa A^k) a^k_0}{\sum_{k=1}^{K} \exp(\kappa A^k)} \tag{6}$$

where κ controls the emphasis on larger advantages. After executing a^*, we observe the next state and repeat until the episode terminates. We list the full algorithm in Algorithm 1 (Appendix B.1).

3.2 Experiments

We compare our proposed SHARP with prior plug-and-play planning methods, MBOP (Argenson and Dulac-Arnold, 2021) and LOOP (Sikchi et al., 2021). Following the recommendation of Jackson et al. (2025), we train Q-functions using the two offline RL algorithms, IQL (Kostrikov et al., 2022) and ReBRAC (Tarasov et al., 2023), and use their policies as the action proposer u. We evaluate on the D4RL locomotion benchmarks (Fu et al., 2020) 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). Details of the environments, datasets, and offline RL setup, including the IQL and ReBRAC training details, are provided in Appendix E.4 and F.1.

To examine the stability of all methods across different planning horizons, we vary the maximum planning horizon H from 1 to 8 and evaluate their effects on the planning performance. Following

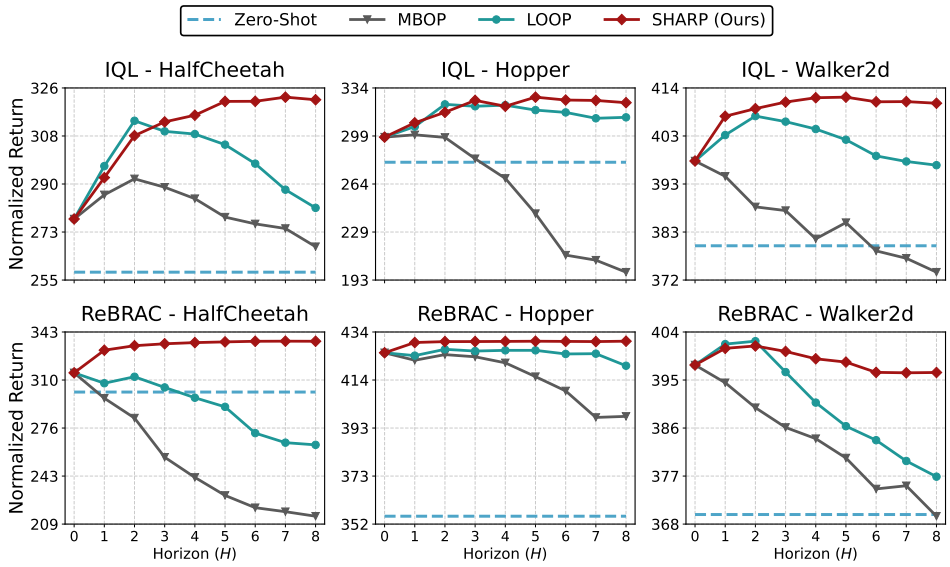


Figure 1: **SHARP is robust to larger horizons.** We vary the test-time maximum planning horizon H for IQL- and ReBRAC-based planners on D4RL locomotion tasks. MBOP and LOOP are sensitive to horizon choice, often improving at short horizons but degrading as H increases due to accumulated model errors. In contrast, SHARP maintains stable performance across horizons.

previous works, 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 1 show that the planning horizon H has a critical yet double-edged effect on 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. Therefore, these methods are highly sensitive to H and require task-specific tuning.

In contrast, our proposed SHARP addresses this limitation through soft-horizon aggregation, enabling a universally large maximum horizon H by weighting deeper planning steps according to uncertainty estimates from ensemble branches, and automatically down-weighting error-prone predictions. As a result, SHARP maintains stable and strong performance as H increases, reducing the need for task-specific horizon tuning. This suggests that H can instead be treated as a computational-budget parameter determined by the available test-time computation rather than through task-specific tuning.

To evaluate planning performance without online hyperparameter tuning, we grid search 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 average performance across all configurations in Table 1. Under this challenging evaluation protocol, our method consistently outperforms both the original offline RL policies (Zero-Shot) and the MBOP and LOOP baselines. The gains across IQL and ReBRAC show that our approach substantially improves performance while reducing sensitivity to online hyperparameter tuning.

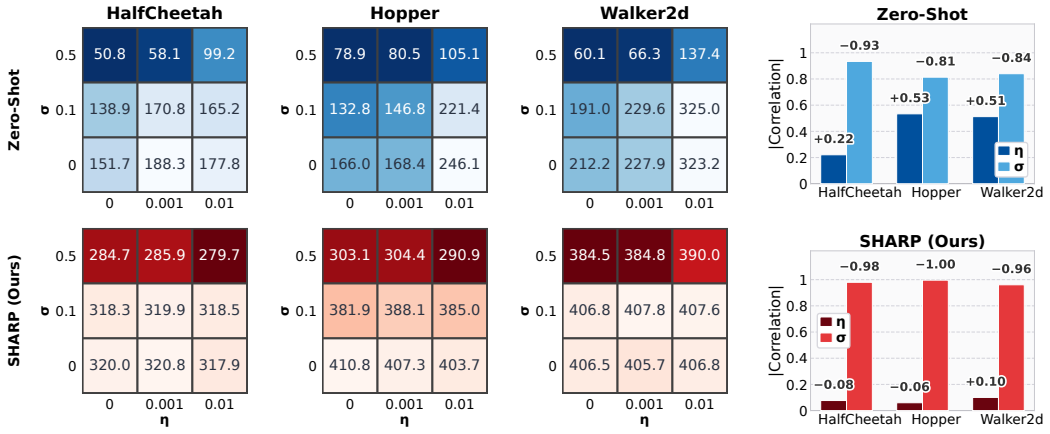
We ablate key design choices of SHARP in Appendix B.2, including hard precision-based horizon selection instead of soft aggregation, and return-based aggregation instead of advantage-based aggregation. Additional statistics, including the best performance after hyperparameter tuning, are reported in Appendix B.3, while sensitivity to other hyperparameters is analyzed in Appendix B.4.

4 Simplifying Policy Extraction via Planning

Having addressed the hyperparameter sensitivity introduced by test-time planning itself, we next investigate whether planning can also reduce sensitivity to offline policy extraction. Plug-and-play planning methods typically use an offline RL policy as the action proposer (Sikchi et al., 2021; Jeong et al., 2025). However, distilling a strong policy from a learned Q-function often requires selecting a policy-extraction objective and tuning its hyperparameters (Park et al., 2024), introducing another

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 inverse temperatures κ, simulating a setting where online selection of the best planning configuration is unavailable. SHARP achieves stronger average performance across IQL and ReBRAC.

Environment	D	IQL				ReBRAC			
		Zero-Shot	MBOP	LOOP	SHARP (Ours)	Zero-Shot	MBOP	LOOP	SHARP (Ours)
HalfCheetah	R	6.7	17.7	19.0	**19.5**	**25.3**	24.6	25.0	24.7
	M	47.2	51.6	53.6	**54.5**	57.4	50.9	56.5	**62.5**
	MR	43.0	43.1	46.4	**47.7**	44.9	41.8	47.8	**51.1**
	ME	89.0	72.5	**92.7**	90.1	95.2	41.3	80.2	**102.9**
	FR	71.6	71.8	74.8	**80.7**	78.4	58.9	71.4	**82.5**
Hopper	R	7.6	7.8	**7.9**	**7.9**	8.6	8.6	8.9	**9.0**
	M	51.8	41.2	56.4	**58.6**	99.7	86.2	96.1	**101.6**
	MR	55.9	70.7	**87.9**	87.0	86.2	95.0	**99.2**	**99.2**
	ME	**70.9**	30.9	34.7	30.4	54.3	86.7	91.2	**107.4**
	FR	**93.4**	69.1	91.6	91.8	106.2	101.9	106.3	**106.6**
Walker2d	R	4.8	4.8	5.1	**5.6**	16.4	7.6	10.0	**17.0**
	M	83.4	80.7	87.6	**89.4**	76.7	79.7	81.2	**82.0**
	MR	84.1	82.5	92.8	**95.4**	72.7	77.0	82.2	**83.2**
	ME	**109.5**	108.2	109.1	108.7	108.4	100.8	108.7	**110.1**
	FR	97.9	96.5	99.9	**100.4**	95.8	95.1	98.3	**101.1**



(a) Normalized Return under different σ and η.

(b) Pearson correlation (r)

Figure 2: **Effect of Behavioral regularization ($\sigma \downarrow$) and Q exploitation ($\eta \uparrow$) on zero-shot and planning performance.** Zero-shot performance varies with both σ and η, whereas SHARP is largely insensitive to η and primarily improves with stronger behavioral regularization, showing that BC is already a sufficient action proposer.

source of sensitivity that is often addressed through online evaluation. In policy extraction, it generally trades off two objectives: behavioral regularization, which keeps the policy close to the offline data distribution, and Q-function exploitation, which favors actions with higher predicted value. While balancing these two objectives has a substantial influence on the zero-shot policy performance, it is unclear whether both are necessary when the policy serves only as an action proposer for planning.

To understand how policy-extraction choices affect planning, we first disentangle the influence of behavioral regularization and Q-function exploitation through a controlled inference-time intervention (Section 4.1), and then verify the resulting observations using standard policy-extraction objectives (Section 4.2). Our results show that behavioral regularization remains important for planning, whereas Q-function exploitation provides little benefit. This makes a simple behavior-cloned policy a sufficient action proposer, removing the need to tune Q-exploitation strength during policy extraction.

4.1 Disentangling Behavioral Regularization and Q Exploitation

To isolate the effects of behavioral regularization and Q-function exploitation for planning, we construct a controlled inference-time policy extraction procedure starting from a behavior-cloning

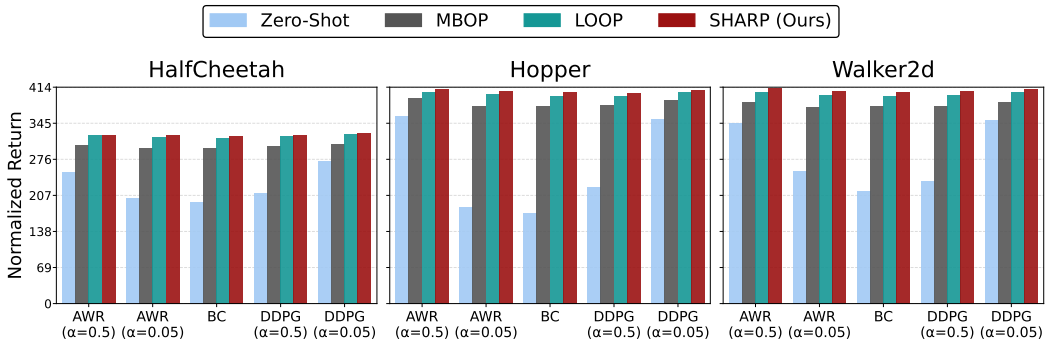


Figure 3: **Standard policy extraction can be simplified for planning.** Although AWR, DDPG+BC, and BC produce substantially different zero-shot policies, they yield similar planning performance, suggesting that choosing a simple BC action proposer can avoid Q-exploitation tuning.

policy. We first perturb an action sampled from the BC policy:

$$a_{\text{reg}} = a_{\text{BC}} + \epsilon, \qquad a_{\text{BC}} \sim \pi_{\text{BC}}(\cdot \mid s), \quad \epsilon \sim \mathcal{N}(0, \sigma I), \tag{7}$$

where increasing σ induces larger deviations from the behavior-cloned distribution, thereby weakening the effective behavioral constraint. We then apply a gradient-ascent step on the learned Q-function,

$$a = a_{\text{reg}} + \eta \nabla_{a_{\text{reg}}} Q(s, a_{\text{reg}}), \tag{8}$$

where η controls the degree of Q-function exploitation (Park et al., 2024). Thus, σ and η provide controlled interventions on behavioral regularization and Q-function exploitation, respectively.

We evaluate both zero-shot policy and planning (SHARP) performance using IQL's value function while varying σ and η of the action independently. The results in Figure 2 reveal a clear distinction between the properties required for zero-shot policy execution and planning. Both behavioral regularization and Q-function exploitation play a significant role in the zero-shot performance. In contrast, while planning remains highly sensitive to weakening the behavioral constraint, it is largely insensitive to Q-function exploitation η. Stronger exploitation of the Q-function provides little additional benefit when the policy serves only as an action proposer for planning.

These results suggest that an action proposer does not need to achieve strong zero-shot performance. Its primary role is to provide plausible candidate actions supported by the offline data, while the planner itself performs the subsequent candidate evaluation. Thus, a behavior-cloned policy may already be sufficient. We next test whether this conclusion continues to hold for policies obtained through standard offline policy extraction methods.

4.2 Behavior Cloning Is a Sufficient Action Proposer

To verify whether the previous conclusion holds for standard offline policy-extraction methods, we compare behavior cloning (BC) with two widely used policy extraction objectives, AWR (Peng et al., 2019) and DDPG+BC (Fujimoto and Gu, 2021). Both introduce a hyperparameter α that controls the trade-off between behavioral regularization and Q-function exploitation, but in opposite directions.

In AWR, a larger α increases Q-function exploitation, whereas in DDPG+BC it strengthens behavioral regularization. Further details of the objectives and training setup are provided in Appendix F.1.

We extract policies with AWR and DDPG+BC with varying values of α from Q functions trained by IQL. We then use each extracted policy as the action proposer for MBOP, LOOP, and SHARP (Ours). To isolate the effect of the action proposer from planner hyperparameter sensitivity, for each planning method we follow prior works (Sikchi et al., 2021; Jeong et al., 2025) and report the best-performing setting averaged over 3 seeds and 20 evaluation episodes by sweeping $H \in \{2, 4\}$ and $\kappa \in \{0.1, 0.5, 1.0, 5.0, 10.0\}$.

As shown in Figure 3, policy-extraction objective and Q-exploitation strength α substantially affect zero-shot policy performance. In contrast, planning performance remains largely unchanged across extraction methods and values of α. In particular, a simple BC policy achieves planning performance comparable to policies with substantially stronger zero-shot performance.

Table 2: **SHARP-BC remains competitive with tuned offline planning baselines using a single planning configuration.** Baseline results use their reported best-performing settings, while SHARP-BC uses the same planning hyperparameters across all datasets.

Environment	Dataset	Sequence-based Planning			Model-based Planning		SHARP-BC (Ours)	
		TT	TAP	DV	MBOP	MOPP	IQL*	ReBRAC*
HalfCheetah	Medium	46.9	45.0	50.4	44.6	44.7	54.3	**55.4**
	Medium-Replay	41.9	40.8	45.8	42.3	43.1	49.0	**50.4**
	Medium-Expert	95.0	91.8	92.7	105.9	**106.2**	99.4	102.3
Hopper	Medium	61.1	63.4	**83.6**	48.8	31.8	67.9	67.8
	Medium-Replay	91.5	87.3	91.9	12.4	32.3	93.7	**96.5**
	Medium-Expert	**110.0**	105.5	**110.0**	55.1	95.4	98.2	108.3
Walker2d	Medium	79.0	64.9	82.8	41.0	80.7	**85.1**	82.3
	Medium-Replay	82.6	66.8	85.0	9.7	18.5	**88.1**	82.6
	Medium-Expert	101.9	107.4	109.2	70.2	92.9	109.8	**110.1**
Locomotion Avg.		78.9	74.8	83.5	47.8	60.6	82.8	**84.0**
Pen	Cloned	11.4	57.4	80.2	63.2	73.2	97.1	**109.3**
	Expert	72.0	127.4	122.2	105.5	149.5	150.4	**150.6**
Hammer	Cloned	0.5	1.2	**11.9**	4.2	4.9	3.1	0.80
	Expert	15.5	127.6	125.8	107.6	128.7	130.3	**131.0**
Door	Cloned	-0.1	**11.7**	1.5	0.0	5.6	1.13	0.77
	Expert	94.1	104.8	104.7	101.2	105.3	104.8	**106.4**
Relocate	Cloned	-0.1	-0.2	**0.6**	0.1	-0.1	-0.20	-0.19
	Expert	10.3	105.8	108.9	41.7	98.0	113.2	**113.6**
Adroit Avg.		25.5	67.0	69.5	52.9	70.6	73.7	**76.5**

These results support the findings in Section 4.1: Q-function exploitation is important for zero-shot execution but provides little additional benefit once the policy is used only as an action proposer. This motivates using behavior cloning as a simple action proposer, avoiding the need to select a policy-extraction objective or tune its Q-exploitation strength through online evaluation.

5 SHARP-BC: ROBUST TEST-TIME PLANNING FOR OFFLINE RL

Combining all the findings, we propose SHARP-BC, which uses soft horizon aggregation together with a simple BC policy as the action proposer. SHARP-BC removes the need to carefully select both the planning horizon and the Q-exploitation strength of the proposer. We examine whether SHARP-BC can (1) retain strong planning performance without task-specific online tuning, and, more importantly, (2) reduce sensitivity to the hyperparameters of the underlying offline RL algorithm.

We first compare SHARP-BC against other planning baselines on D4RL (Fu et al., 2020) locomotion and Adroit dexterous manipulation tasks, including sequence-based methods (TT; Janner et al. 2021, TAP; zhengyao jiang et al. 2023, DV; Lu et al. 2025) and model-based approaches (MBOP; Argenson and Dulac-Arnold 2021, MOPP; Zhan et al. 2022). We report SHARP-BC using a single set of planning hyperparameters shared across all datasets, without task-specific tuning. In contrast, the baseline methods report results using their best-performing hyperparameters obtained through online hyperparameter search. The results in Table 2 show that, even without any tuning, SHARP-BC consistently achieves competitive or superior performance relative to these fully tuned baselines, showing that reduced tuning does not come at the cost of planning quality. We include the descriptions and hyperparameters of each method in Appendix E.3.

More importantly, we examine whether SHARP-BC can reduce sensitivity to the hyperparameters of the underlying offline RL method. In Figure 4, we compare zero-shot policies and SHARP-BC using Q-functions obtained from various IQL and ReBRAC configurations on locomotion tasks. The results show that, while zero-shot performance varies substantially across hyperparameters, SHARP-BC remains much closer to the best attainable performance across configurations. We further evaluate this robustness on additional datasets, with full training configurations and hyperparameter details in Appendix D.

These results conclude that SHARP-BC not only reduces tuning within the planner, but also makes deployed performance less sensitive to the underlying offline RL hyperparameters.

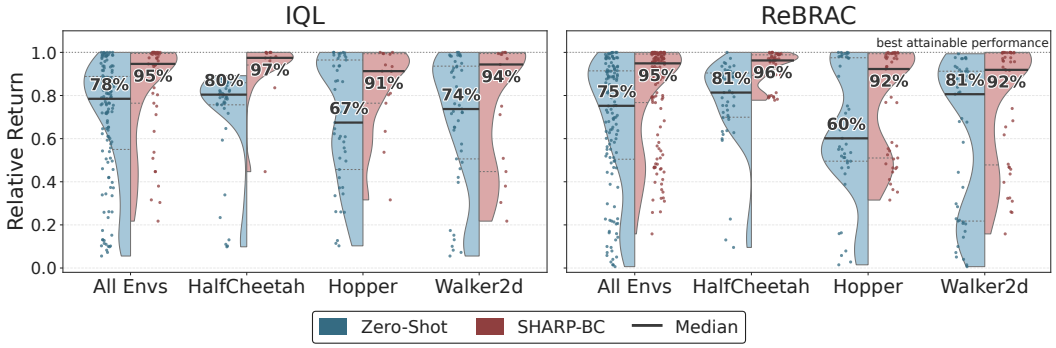


Figure 4: **SHARP-BC reduces sensitivity to underlying offline RL hyperparameters.** Across various IQL and ReBRAC configurations, zero-shot performance varies substantially, while SHARP-BC remains consistently closer to the best attainable performance.

6 Related work

Offline RL learns policies from static datasets without online interaction (Levine et al., 2020). 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 (Kumar et al., 2020; Cheng et al., 2022), behavior regularization (Wu et al., 2019; Fujimoto and Gu, 2021; Tarasov et al., 2023), in-sample learning (Kostrikov et al., 2022; Xu et al., 2023; Garg et al., 2023), and uncertainty-aware regularization (An et al., 2021; Ghasemipour et al., 2022). Offline RL's performance can be highly sensitive to policy extraction (Fu et al., 2022; Park et al., 2024) and its hyperparameters (Jackson et al., 2025; Kurenkov and Kolesnikov, 2022). Our work studies whether offline model-based planning can reduce this sensitivity.

Offline model-based planning learns dynamics and reward models from offline data for test-time policy improvement. MBOP (Argenson and Dulac-Arnold, 2021) is an early work to introduce planning into the offline setting, while MOPP (Zhan et al., 2022) prunes uncertain branches for robustness, and IQL-TD-MPC (Chitnis et al., 2024) combines TD-MPC with IQL (Kostrikov et al., 2022) to address limited offline data coverage. Plug-and-play methods further leverage planning to improve on top of existing policy and value models. LOOP-offline (Sikchi et al., 2021) improves robustness by ensembling dynamics models and penalizing the returns. While RefPlan (Jeong et al., 2025) improves generalization through dynamics modeling with VariBAD (Zintgraf et al., 2020), we instead focus on mitigating hyperparameter sensitivity at the planning-algorithm level.

Uncertainty-aware model-based RL uses uncertainty estimates to mitigate errors in learned-dynamics rollouts during offline policy training. Existing approaches include penalty-based methods, which penalize model-generated rewards or Bellman targets according to estimated uncertainty (Yu et al., 2020; Sun et al., 2023); truncation-based methods, which terminate rollouts above an uncertainty threshold (Kidambi et al., 2020; Ni et al., 2026); and uncertainty-weighted value-expansion methods, which combine targets across rollout horizons by estimated reliability (Buckman et al., 2018; Jeong et al., 2023). These mechanisms commonly introduce new uncertainty-specific hyperparameters whose optimal values vary across tasks (Jackson et al., 2025). In contrast, SHARP-BC uses uncertainty for test-time planning, reducing sensitivity to the maximum horizon without additional uncertainty-specific hyperparameters. We further discuss this relationship in Appendix B.5.

7 Conclusions

This work aims to improve the performance and reduce the hyperparameter sensitivity of offline reinforcement learning through plug-and-play test-time planning. To this end, we propose SHARP-BC, which combines uncertainty-aware aggregation over planning horizons with a simple behavior-cloned action proposer. SHARP-BC achieves strong offline planning performance without planning-specific hyperparameter tuning while substantially reducing sensitivity to the underlying offline RL method's hyperparameters. Overall, our results show that test-time planning can improve not only performance, but also the robustness of offline RL.

AI USE STATEMENT

In this work, we used generative AI tools for "implement methods, support qualitative and thematic data analysis" . We have not used generative AI tools for "design or provide feedback on research methodology or experiments, generate synthetic data sets, help develop theoretical models or conceptual frameworks, assist with translation, clean and reformat datasets, interpret results" , and "formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses" are not applicable to this work. Additionally, we used generative AI tools for "create or modify scientific figures or images, create or edit software code, summarize or analyze existing literature, brainstorming, sourcing/searching for information, edit a research paper to improve readability, identify relevant literature, suggest a structure for a research paper, propose a title or keywords for a research paper" . We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness by the authors. AI-assisted literature searches and summaries were checked against the original sources, and all cited references were verified by the authors. AI-generated plotting code was reviewed and verified by the authors to ensure that the resulting figures accurately represent the underlying data and results. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed information to facilitate reproduction of our results. The SHARP planning procedure is described in Section 3.1, with complete pseudocode provided in Appendix B.1. Experimental settings for evaluating robustness to offline RL hyperparameters are described in Appendix D. Appendix E documents the evaluated tasks and datasets, baseline methods, and their planning configurations. Appendix F.1 provides implementation and training details for the underlying offline RL algorithms, behavior-cloned and policy-extraction action proposers, and the learned dynamics and reward models, including network architectures and hyperparameters. We report results over multiple random seeds and specify the evaluation protocols alongside the corresponding experiments. Together, these details specify the algorithms, training procedures, evaluation settings, and hyperparameters used throughout our experiments.

REFERENCES

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.

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.

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.

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.

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

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

Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*, 2020.

Vladislav Kurenkov and Sergey Kolesnikov. Showing your offline reinforcement learning work: Online evaluation budget matters. In *International Conference on Machine Learning*, 2022.

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.

Matthew Thomas Jackson, Uljad Berdica, Jarek Liesen, Shimon Whiteson, and Jakob Nicolaus Foerster. A clean slate for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2025.

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

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.

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

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

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.

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.

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

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.

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.

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

Grady Williams, Andrew Aldrich, and Evangelos A. Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40, 2017.

Tianwei Ni, Esther Derman, Vineet Jain, Vincent Taboga, Siamak Ravanbakhsh, and Pierre-Luc Bacon. Long-horizon model-based offline reinforcement learning without explicit conservatism. In *International Conference on Machine Learning*, 2026.

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.

Jihwan Jeong, Xiaoyu Wang, Michael Gimelfarb, Hyunwoo Kim, Baher abdulhai, and Scott Sanner. Conservative bayesian model-based value expansion for offline policy optimization. In *The Eleventh International Conference on Learning Representations*, 2023.

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

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.

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

zhengyao jiang, Tianjun Zhang, Michael Janner, Yueying Li, Tim Rocktäschel, Edward Grefenstette, and Yuandong Tian. Efficient planning in a compact latent action space. In *The Eleventh International Conference on Learning Representations*, 2023.

Haofei Lu, Dongqi Han, Yifei Shen, and Dongsheng Li. What makes a good diffusion planner for decision making? In *The Thirteenth International Conference on Learning Representations*, 2025.

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

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.

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

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.

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.

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.

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.

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

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.

Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. In *International Conference on Learning Representations*, 2020.

Yihao Sun, Jiaji Zhang, Chengxing Jia, Haoxin Lin, Junyin Ye, and Yang Yu. Model-Bellman inconsistency for model-based offline reinforcement learning. In *International Conference on Machine Learning*, 2023.

David Brandfonbrener, William F Whitney, Rajesh Ranganath, and Joan Bruna. Offline RL without off-policy evaluation. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

Kerong Wang, Hanye Zhao, Xufang Luo, Kan Ren, Weinan Zhang, and Dongsheng Li. Bootstrapped transformer for offline reinforcement learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

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 **13**

B Method Details and Extended Experiments of SHARP **14**
- B.1 SHARP Algorithm . . . 14
- B.2 Ablation studies of SHARP . . . 14
- B.3 Additional Statistics . . . 15
- B.4 Computational budget and model capacity . . . 16
- B.5 Transfer uncertainty-aware MBRL methods to planning . . . 18
- B.6 Inverse-temperature parameter (κ) . . . 19

C Generative Action Proposer **20**

D SHARP-BC Robustness to Offline RL Hyperparameters **20**

E Baselines and dataset descriptions **21**
- E.1 Offline RL baselines . . . 21
- E.2 Plug-and-play planning methods . . . 22
- E.3 Offline planning baselines . . . 23
- E.4 Tasks and datasets . . . 24

F Implementation details **25**
- F.1 BC and Offline RL . . . 25
- F.2 Plug-and-play planning . . . 26

G Experiments compute resources **26**

A LIMITATIONS

The broader vision of this work is to use test-time planning not only to improve the performance of offline RL, but also to make its deployed performance more robust to choices made during offline training. SHARP-BC takes a step toward this goal by reducing sensitivity to both planning and offline RL hyperparameters; however, we do not claim that it eliminates all sources of sensitivity in offline RL. We discuss several remaining limitations and directions toward this broader goal below.

Dependence on the underlying offline RL. Plug-and-play planning learned components from the underlying offline RL algorithm. While our experiments show that SHARP-BC can substantially reduce variation in deployed performance across different IQL and ReBRAC hyperparameter configurations, it can't recover strong performance when these configurations consistently produce unreliable value estimates. This limitation is evident on several Adroit cloned datasets, where both zero-shot execution and SHARP-BC remain near-degenerate across configurations (Appendix D). Thus, SHARP-BC should be understood as making performance more robust among sufficiently informative underlying models, rather than correcting arbitrary failures of offline RL. Developing planning methods that can detect or remain effective under severely inaccurate value functions is an important direction for future work.

Inverse-temperature parameter κ. SHARP removes the need to carefully select a planning horizon, but still retains the inverse-temperature parameter κ used by the softmax action-selection rule. As discussed further in Appendix B.6, κ can affect planning performance, and our method therefore does not completely eliminate planning hyperparameters. In our evaluations, we avoid relying on task-specific selection of κ: we report performance averaged across the evaluated H and κ configurations when studying tuning sensitivity, and use a single fixed planning configuration across tasks for SHARP-BC. Developing an effective action-selection mechanism that removes this remaining parameter is left for future work.

Generative action proposers. Our main analysis focuses on standard non-generative policy parameterizations. Generative policies, such as flow- or diffusion-based policies, may capture multimodal behavior distributions and could interact differently with test-time planning. We provide an initial study with a flow-based policy in Appendix C, but a systematic investigation of generative action proposers is beyond the scope of this work.

B Method Details and Extended Experiments of SHARP

B.1 SHARP Algorithm

Algorithm 1 provides the full SHARP procedure. At each environment step, SHARP simulates diverse future trajectories using an ensemble of learned dynamics models. It then estimates the horizon-specific advantage and precision, combines them through soft horizon aggregation, and selects the final action based on the resulting aggregated advantages.

Algorithm 1 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, inverse temperature κ
Observe initial state s_0
while episode not terminated **do**
  for $k = 1$ **to** K **do**
    $a_0^k \sim u(\cdot \mid s_0)$
    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 $h = 1$ **to** H **do**
          Sample next state $\hat{s}_{h,e,p}^k \sim \hat{T}_e(\cdot \mid \hat{s}_{h-1,e,p}^k, a_{h-1,e,p}^k)$
          Sample next action $a_{h,e,p}^k \sim u(\cdot \mid \hat{s}_{h,e,p}^k)$
          $G_{h,e,p}^k = \sum_{t=0}^{h-1} \gamma^t \hat{r}(\hat{s}_{t,e,p}^k, a_{t,e,p}^k) + \gamma^h \hat{Q}(\hat{s}_{h,e,p}^k, a_{h,e,p}^k)$
        end for
      end for
    end for
  end for
  for $h = 1$ **to** H **do**
    $B_h = \mathrm{mean}_{j,e,p}(G_{h,e,p}^j)$ ▷ Calculate a shared baseline for each horizon
    $A_{h,e,p}^k = G_{h,e,p}^k - B_h, \quad \forall k, e, p$ ▷ Substract the baseline to get advantage
    $\mu_h^k = \mathrm{mean}_{e,p}(A_{h,e,p}^k) \quad \forall k$ ▷ Calculate horizon-specific advantage
    $\rho_h^k = 1/\mathrm{var}_{e,p}(A_{h,e,p}^k) \quad \forall k$ ▷ Calculate horizon-specific precision
  end for
  $A^k = \sum_{h=1}^{H} \left(\frac{\rho_h^k}{\sum_{h'=1}^{H} \rho_{h'}^k} \right) \mu_h^k$ ▷ Compute A^k through soft horizon aggregation
  Select $a^* = \frac{\sum_{k=1}^{K} \exp(\kappa A^k) a_0^k}{\sum_{k=1}^{K} \exp(\kappa A^k)}$ ▷ Select action via advantage softmax
  Execute a^*, observe real next state $s_0 \leftarrow s'$
end while

B.2 Ablation studies of SHARP

In this section, we ablate the two core design choices of SHARP: weighted advantage aggregation and soft horizon aggregation.

First, we investigate a **weighted return** aggregation variant as an alternative to our proposed **weighted advantage** aggregation. Instead of calculating the mean and variance using advantages across candidates, it calculates them directly from the return. Let $G_{h,e,p}^k$ denote the estimated return for

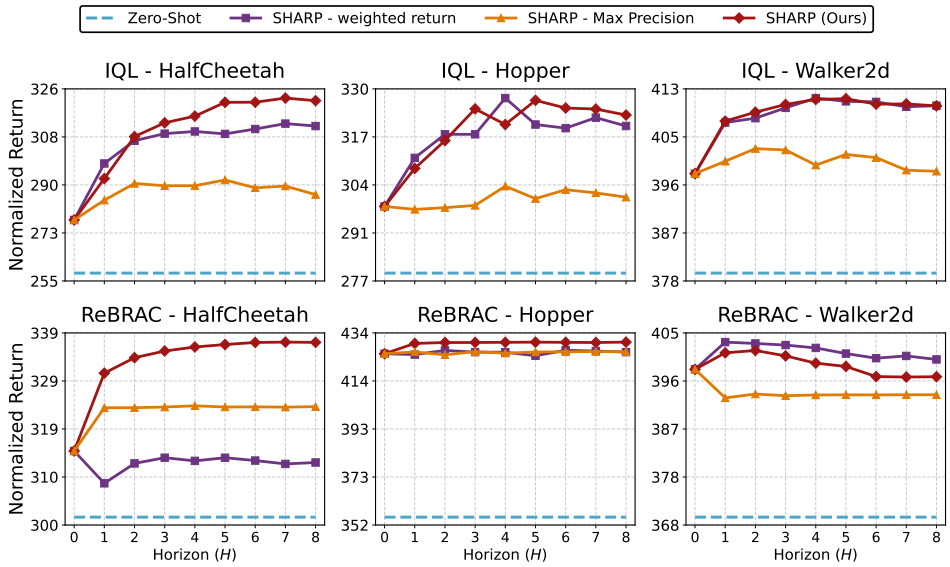


Figure 5: **Ablation of weighted advantage and soft horizon aggregation.** We ablate the weighted advantage into return advantage and soft horizon aggregation into choosing a horizon with max precision. The results show that our proposed SHARP matches or outperforms these variants.

candidate k at horizon h, ensemble member e, and particle p. The mean and precision of the estimated returns at each horizon are computed as

$$\mu_h^k = \text{mean}_{e,p}(G_{h,e,p}^k), \quad \rho_h^k = 1/\text{var}_{e,p}(G_{h,e,p}^k). \tag{9}$$

The aggregated return for each candidate is then obtained by taking a precision-weighted average across all evaluated horizons: $G^k = \sum_{h=1}^{H} \left(\frac{\rho_h^k}{\sum_{h'=1}^{H} \rho_{h'}^k} \right) \mu_h^k$, where the action to be executed in the environment is $a^* = \frac{\sum_{k=1}^{K} \exp(\kappa G^k) a_0^k}{\sum_{k=1}^{K} \exp(\kappa G^k)}$.

As shown in Figure 5, weighted advantage aggregation matches or outperforms weighted return aggregation. Model-based planning aims to select which candidate action to execute, focusing on relative performance to other candidates rather than absolute return values. Subtracting the horizon-specific baseline prevents the horizon with higher overall estimated returns from dominating others, allowing the aggregation to reflect their estimated relative performance across horizons.

Next, we consider a **Max Precision** approach instead of the **soft horizon aggregation** we use. Specifically, rather than computing a softly weighted sum across all evaluated horizons, it selects the expected advantages from the single horizon h that exhibits the highest precision (lowest variance). Let $h_k = \arg\max_{h \in \{1,\dots,H\}} \rho_h^k$, where ρ_h^k denotes the precision of the advantage estimate at horizon h. Then the resulting advantage for the Max Precision approach is $A^k = \mu_{h_k}^k$. As shown in Figure 5, the Max Precision variant substantially underperforms our proposed soft aggregation method. This performance gap suggests that relying exclusively on the most precise horizon discards valuable information from other rollout depths, highlighting the importance of aggregating advantage estimates across multiple horizons.

B.3 Additional Statistics

In Section 3.2 (Table 1), we evaluate robustness by averaging performance over all tested planning horizons H and action-selection inverse temperatures κ. Table 3 provides the corresponding mean performance together with the standard deviation across three random seeds. This evaluation reflects the setting of primary interest in this work, where online evaluation is unavailable for selecting a task-specific planning configuration.

To complement this tuning-free evaluation, we also report the best performance over the same H and κ search space in Table 4. This represents an oracle setting in which online evaluation is available to

select the best planning hyperparameters. Even under this favorable setting for the baselines, SHARP remains competitive with or outperforms MBOP and LOOP across the evaluated tasks. These results indicate that the robustness gained through soft horizon aggregation does not come at the cost of best-case planning performance.

Table 3: **Planning performance without hyperparameter selection.** We report the mean normalized return across all evaluated settings for each planning method.

Environment	Dataset	IQL				ReBRAC			
		Zero-shot	MBOP	LOOP	SHARP (Ours)	Zero-shot	MBOP	LOOP	SHARP (Ours)
HalfCheetah	R	6.7±2.9	17.7±0.7	19.0±0.9	**19.5±1.1**	**25.3±0.8**	24.6±0.8	25.0±0.8	24.7±0.7
	M	47.2±0.1	51.6±0.2	53.6±0.1	**54.5±0.2**	57.4±0.8	50.9±1.2	56.5±0.3	**62.5±0.6**
	MR	43.0±0.3	43.1±0.5	46.4±0.2	**47.7±0.2**	44.9±0.4	41.8±1.0	47.8±0.3	**51.1±0.2**
	ME	89.0±2.5	72.5±3.2	**92.7±1.2**	90.1±2.7	95.2±0.6	41.3±2.8	80.2±0.8	**102.9±0.6**
	FR	71.6±0.4	71.8±1.3	74.8±1.0	**80.7±1.1**	78.4±1.4	58.9±1.6	71.4±0.4	**82.5±1.0**
Hopper	R	7.6±0.2	7.8±0.2	**7.9±0.3**	**7.9±0.3**	8.6±1.0	8.6±1.2	8.9±1.4	**9.0±1.5**
	M	51.8±2.0	41.2±2.3	56.4±4.0	**58.6±5.8**	99.7±0.4	86.2±4.8	96.1±3.7	**101.6±0.2**
	MR	55.9±12.4	70.7±3.0	**87.9±1.2**	87.0±0.9	86.2±7.2	95.0±1.6	**99.2±0.5**	**99.2±0.5**
	ME	**70.9±18.4**	30.9±2.8	34.7±3.5	30.4±3.8	54.3±1.3	86.7±8.4	91.2±8.6	**107.4±2.2**
	FR	**93.4±9.8**	69.1±6.1	91.6±2.9	91.8±2.7	106.2±1.0	101.9±2.0	106.3±0.3	**106.6±0.4**
Walker2d	R	4.8±1.0	4.8±0.6	5.1±0.5	**5.6±0.6**	16.4±6.9	7.6±2.5	10.0±3.9	**17.0±6.1**
	M	83.4±2.0	80.7±3.0	87.6±1.7	**89.4±1.5**	76.7±1.3	79.7±1.3	81.2±0.5	**82.0±0.4**
	MR	84.1±2.0	82.5±3.5	92.8±2.0	**95.4±1.9**	72.7±1.6	77.0±2.6	82.2±2.0	**83.2±1.7**
	ME	**109.5±0.2**	108.2±0.2	109.1±0.2	108.7±0.2	108.4±0.2	100.8±2.2	108.7±0.4	**110.1±0.2**
	FR	97.9±0.7	96.5±1.5	99.9±0.6	**100.4±0.5**	95.8±1.3	95.1±1.6	98.3±0.8	**101.1±0.7**

Table 4: **Best planning performance after hyperparameter selection.** We report the best normalized return across all evaluated settings for each planning method.

Environment	Dataset	IQL				ReBRAC			
		Zero-shot	MBOP	LOOP	SHARP (Ours)	Zero-shot	MBOP	LOOP	SHARP (Ours)
HalfCheetah	R	6.7±2.9	21.4±0.6	23.2±0.8	**25.2±1.0**	25.3±0.8	25.6±0.9	**26.5±0.6**	25.8±0.4
	M	47.2±0.1	55.3±0.2	58.4±0.4	**60.5±0.3**	57.4±0.8	59.3±0.2	62.1±1.0	**66.4±0.5**
	MR	43.0±0.3	45.5±0.1	48.9±0.2	**52.1±0.5**	44.9±0.4	48.7±0.3	51.8±0.5	**54.4±0.3**
	ME	89.0±2.5	91.1±1.7	**101.5±1.0**	101.4±2.2	95.2±0.6	87.7±5.5	97.0±1.3	**105.9±0.9**
	FR	71.6±0.4	79.3±1.3	83.6±1.1	**84.1±0.9**	78.4±1.4	77.7±0.3	77.8±0.5	**85.2±1.2**
Hopper	R	7.6±0.2	7.9±0.2	8.2±0.4	**8.3±0.4**	8.6±1.0	8.8±1.1	9.1±1.4	**9.3±1.6**
	M	51.8±2.0	53.4±3.3	73.8±3.7	**79.5±5.9**	99.7±0.4	101.4±0.2	102.4±0.2	**103.3±0.1**
	MR	55.9±12.4	98.7±0.7	**99.1±0.2**	98.8±0.2	86.2±7.2	99.6±0.4	**100.0±0.5**	99.7±0.7
	ME	**70.9±18.4**	39.3±3.1	41.7±8.9	37.2±1.6	54.3±1.3	108.7±1.3	108.3±0.4	**110.7±0.5**
	FR	93.4±9.8	102.7±1.2	**106.0±0.1**	104.9±0.5	106.2±1.0	106.3±0.1	**107.4±0.2**	**107.4±0.2**
Walker2d	R	4.8±1.0	5.7±0.7	**6.5±0.6**	6.4±0.5	16.4±6.9	16.3±7.5	19.2±1.5	**20.2±1.2**
	M	83.4±2.0	87.0±2.0	92.4±0.3	**93.5±0.5**	76.7±1.3	83.3±0.5	**84.5±0.6**	83.3±0.6
	MR	84.1±2.0	93.1±1.3	97.2±1.1	**99.4±1.8**	72.7±1.6	83.8±0.9	**87.2±3.2**	84.2±2.8
	ME	109.5±0.2	108.9±0.1	**110.1±0.4**	109.8±0.4	108.4±0.2	110.0±0.2	**111.5±0.5**	110.9±0.3
	FR	97.9±0.7	99.9±1.8	103.5±0.3	**103.7±0.1**	95.8±1.3	101.4±0.7	**104.0±0.8**	103.5±0.5

B.4 COMPUTATIONAL BUDGET AND MODEL CAPACITY

Beyond the planning horizon H studied in Section 3.2, planning also contains parameters that control the available computational budget and dynamics-model capacity. Here, we study how SHARP's performance changes as these resource-related parameters vary.

We first vary the number of candidate action samples K and rollout particles P, which directly control the amount of test-time computation used by the planner. As shown in Figure 6, increasing K or P generally maintains or slightly improves performance. Thus, K and P can be treated primarily as computational-budget parameters and chosen according to the available test-time resources, rather than through task-specific online tuning.

We next vary the ensemble size E and hidden dimension dim of the dynamics model. As shown in Figure 7, planning performance remains largely consistent across ensemble sizes, while increasing the hidden dimension generally improves performance before reaching similar performance at larger capacities.

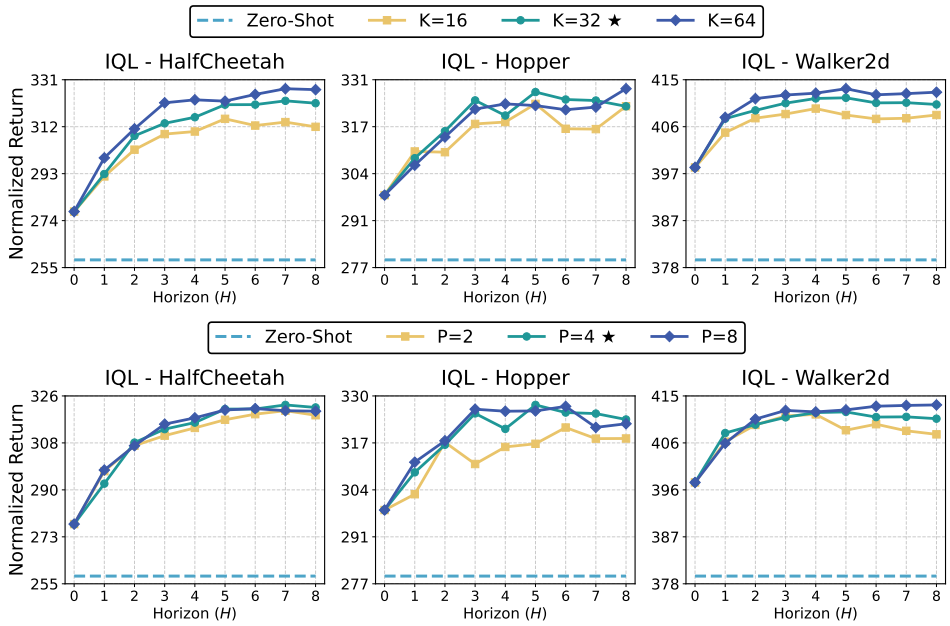


Figure 6: **SHARP scales with computational budget.** We evaluate SHARP's performance on various numbers of candidate action samples K and rollout particles P. Larger K and P generally improve performance, while SHARP remains robust to these choices. $\star$ indicates the configuration used in the main experiments.

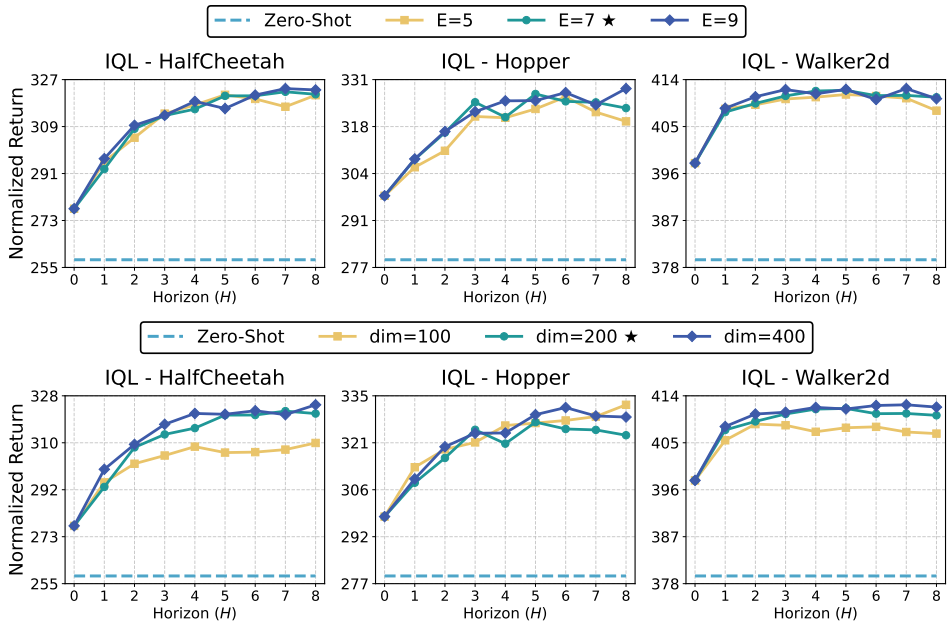


Figure 7: **Effect of dynamics-model capacity on planning performance.** We evaluate SHARP's performance on various ensemble sizes E and hidden dimension sizes dim. SHARP performs consistently across all tested ensemble sizes, showing limited sensitivity to E, while increasing the hidden dimension generally improves performance, with similar performance observed at larger dimensions. $\star$ indicates the configuration used in the main experiments.

These results suggest that these hyperparameters can be selected based on the available computational resources rather than task-specific online tuning.

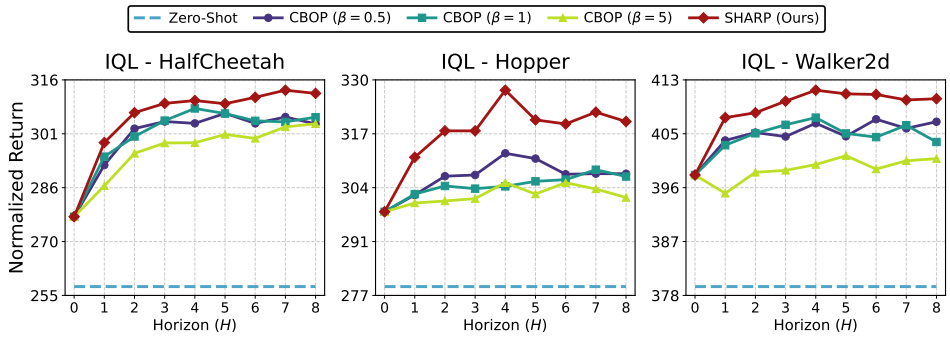


Figure 8: **Performance of the CBOP-planning variant.** Performance comparison across different penalty coefficients β. CBOP-planning variant is sensitive to β and consistently underperforms SHARP. In contrast, SHARP maintains strong and stable performance across planning horizons without requiring an additional uncertainty-related hyperparameter.

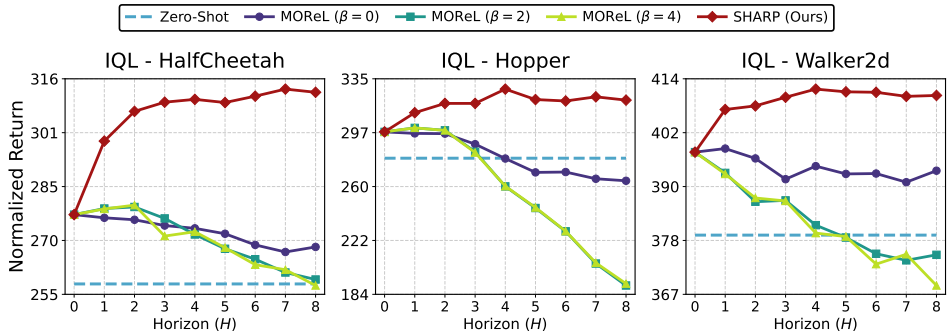


Figure 9: **Performance of the MOReL-planning variant.** Performance comparison across different uncertainty thresholds β. MOReL-planning variant is sensitive to the planning horizon H and the choice of β, often improving at short horizons but degrading substantially as the horizon increases. In contrast, SHARP maintains strong and stable performance across planning horizons without introducing an additional uncertainty-related hyperparameter.

B.5 Transfer uncertainty-aware MBRL methods to planning

In this section, we compare SHARP against planning methods adapted from uncertainty-aware model-based reinforcement learning (MBRL). MBRL and model-based planning both rely on rollouts from learned dynamics models, differing mainly in when and how they use these rollouts: MBRL uses them during training for policy improvement, whereas planning uses them at test time to evaluate candidate actions and guide online decision-making. This close connection naturally raises whether techniques developed for MBRL can also be adapted to test-time planning.

In particular, some uncertainty-aware MBRL methods estimate uncertainty to mitigate model errors. These include: penalizing the model-generated rewards or Bellman targets according to estimated uncertainty (Yu et al., 2020; Sun et al., 2023), terminating rollouts above an uncertainty threshold (Kidambi et al., 2020; Ni et al., 2026), and combining targets across rollout horizons by estimated reliability (Buckman et al., 2018; Jeong et al., 2023). Here we compare two offline MBRL approaches: CBOP (Brandfonbrener et al., 2021), which computes targets across horizons and penalizes them with estimated uncertainty, and MOReL (Kidambi et al., 2020), which terminates rollouts when uncertainty exceeds a threshold.

CBOP uses an ensemble of dynamic models to simulate different future rollouts for better target estimation. It computes the return $G_{h,e} = \sum_{t=0}^{h-1} \gamma^t \hat{r}(\hat{s}_{t,e}, a_{t,e}) + \gamma^h \hat{Q}(\hat{s}_{h,e}, a_{h,e})$ for all h and calculate the mean and precision by $\mu_h = \text{mean}_e(G_{h,e})$, $\rho_h = 1/\text{var}_e(G_{h,e})$. The return is computed

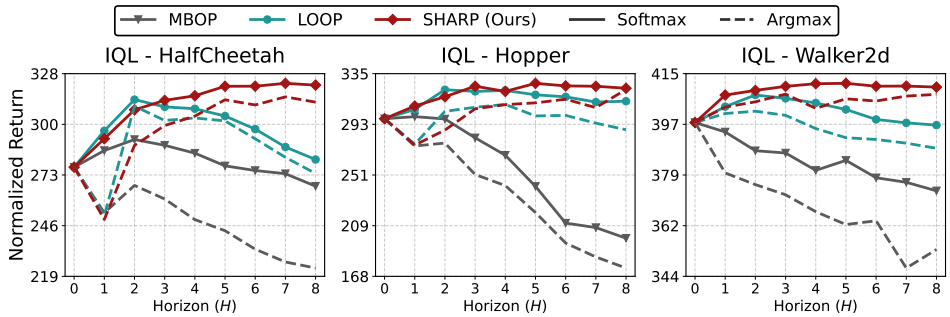


Figure 10: **Removing κ through argmax action selection degrades planning performance.** We compare the soft, temperature-weighted action selection with a parameter-free argmax rule. Across the evaluated domains, argmax generally yields weaker returns, showing that simply removing κ comes at a performance cost.

as

$$G = \sum_{h=1}^{H} \left(\frac{\rho_h}{\sum_{h'=1}^{H} \rho_{h'}} \right) \mu_h - \beta / \sqrt{\sum_{h=1}^{H} \rho_h} \tag{10}$$

where β is the penalty coefficients. Here we adopt its return estimation for model-based planning, and vary the coefficients $\beta \in \{0.5, 1.0, 5.0\}$.

For MOReL, we adapt its pessimistic rollout-termination mechanism to model-based planning. Specifically, we quantify ensemble disagreement as

$$\text{disc}(s, a) = \max_{i \neq j} \|\hat{T}_{e_i}(s, a) - \hat{T}_{e_j}(s, a)\|_2 \tag{11}$$

where $\hat{T}_{e_i}$ and $\hat{T}_{e_j}$ are members of the learned dynamics ensemble. A rollout trajectory is terminated if the disagreement is larger than $\mu_d + \beta\sigma_d$, where μ_d and σ_d denote the empirical mean and standard deviation of disagreement computed over the offline dataset. We evaluate three threshold regimes, $\beta \in \{0,\ 0.5\beta_{\max},\ \beta_{\max}\}$, with $\beta_{\max} = (m_d - \mu_d)/\sigma_d$.

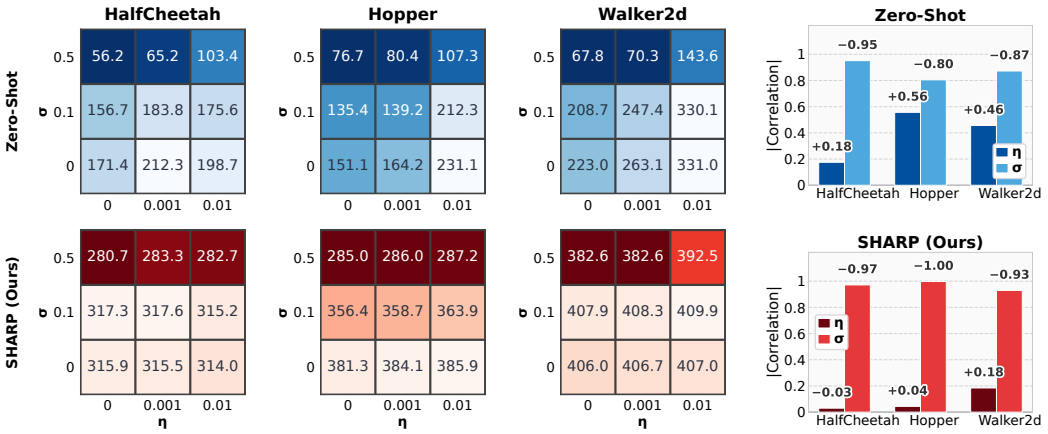
As shown in Figures 8 and 9, the MBRL-planning variants consistently underperform SHARP and are highly sensitive to the penalty coefficient or threshold β. In contrast, across varying planning horizons, SHARP maintains superior, stable planning performance without introducing an additional uncertainty-specific hyperparameter.

B.6 Inverse-temperature parameter (κ)

SHARP removes the need to online-tune the planning horizon H carefully, but retains the inverse-temperature parameter κ used by the soft action-selection rule. This parameter is not exclusive to our proposed method; rather, it is inherited from the MPPI-style action-selection mechanism and shared by previous planning baselines (Argenson and Dulac-Arnold, 2021; Sikchi et al., 2021; Jeong et al., 2025), controlling how strongly candidate actions with larger estimated advantages are emphasized.

One possible way to remove κ is to replace soft action selection with a parameter-free argmax rule that selects only the highest-scoring candidate. We examine this alternative in Figure 10. Across all the evaluated domains, argmax generally produces weaker planning performance than soft action selection. This highlights the reason that prior model-based planning methods, which primarily optimize for performance rather than robustness to hyperparameter choice, typically retain κ and select it through online hyperparameter search.

We thus keep κ in SHARP, which remains a limitation of the current method. Importantly, however, we report results without task-specific tuning of κ (Table 1 and 2). Despite absorbing any performance cost associated with fixing κ, SHARP-BC remains competitive with tuned offline planning baselines (Table 2) while substantially improving robustness to offline RL hyperparameter choices (Figure 4). Therefore, the limitation is that SHARP could potentially perform even better with task-specific tuning of κ, not that such tuning is necessary for the method to retain strong and robust performance.



(a) Normalized Return under different σ and η.

(b) Pearson correlation.

Figure 11: **The policy-extraction finding extends to a flow-based generative action proposer.** Replacing Gaussian BC with a conditional rectified-flow policy yields the same qualitative trend: zero-shot performance benefits from Q-function exploitation, whereas SHARP gains little from additional exploitation and remains primarily dependent on behavioral regularization. This takes a step toward establishing that our observation generalizes beyond a specific policy parameterization.

An interesting direction for future work is to develop an action-selection mechanism that removes this remaining parameter while retaining, or further improving, the performance benefits of soft action selection.

C Generative Action Proposer

Our finding in Section 4 suggests a natural decomposition: the proposer primarily generates plausible, behavior-supported actions, while test-time planning performs policy improvement. Generative policies are therefore potentially suitable action proposers because they can model complex and multimodal behavior distributions.

As an initial study, we repeat the experiment in Section 4.1 while replacing the Gaussian behavior-cloning policy π_{BC} in Equation 7 with a flow-based behavior-cloning policy. Specifically, the flow policy uses conditional rectified flow to transform Gaussian noise into dataset actions conditioned on the current observation. The results in Figure 11 show that the same qualitative conclusion holds: zero-shot performance benefits from Q-function exploitation, while planning gains little from additional exploitation and remains primarily dependent on behavioral regularization.

This suggests that our main observation is not specific to the Gaussian BC parameterization. More expressive generative policies may provide richer candidate actions for planning, making their integration with test-time planning an interesting direction for future work.

D SHARP-BC Robustness to Offline RL Hyperparameters

In Figure 4, we evaluate whether the robustness of SHARP-BC is maintained across different hyperparameter settings for IQL and ReBRAC on locomotion. Here, we provide the experimental details. For each offline RL hyperparameter configuration, we evaluate both the zero-shot policy performance and the planning performance with SHARP-BC, and plot the distribution of their performance across hyperparameter configurations. To make performance comparable across tasks with different return scales, we normalize the return for each task by its best attainable performance, defined as the highest performance achieved across all evaluated hyperparameter configurations and methods (zero-shot or SHARP-BC) for that task. Across all tasks, we consider 9 hyperparameter configurations for each value-function training method: for IQL, the expectile parameter $\tau \in \{0.5, 0.7, 0.9\}$ and AWR coefficient $\beta \in \{0.5, 3, 10\}$; and for ReBRAC, the actor and critic regularization coefficients

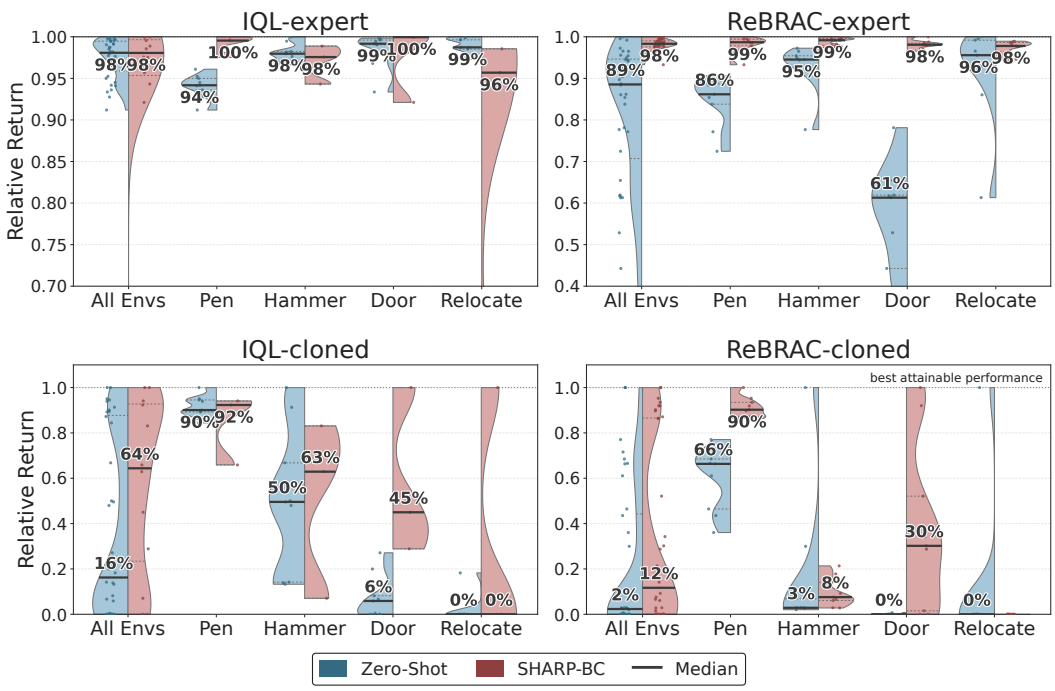


Figure 12: **Robustness to underlying offline RL hyperparameters on Adroit tasks.** We compare zero-shot policy and SHARP-BC across different hyperparameter configurations of IQL and ReBRAC. On expert datasets, SHARP-BC consistently brings performance closer to the best attainable performance across different hyperparameter configurations of IQL and ReBRAC . On the cloned datasets, Hammer, Door, and Relocate exhibit uniformly low absolute performance, preventing a meaningful assessment of hyperparameter robustness; Pen is the exception, where SHARP-BC improves the median performance and reduces variation.

$\beta_1, \beta_2 \in \{0.001, 0.01, 0.1\}$. Each configuration is trained with three random seeds. We fix $H = 8$ and $\kappa = 10$ for planning across all configurations.

We further extend this experiment to the Adroit benchmarks in Figure 12 On the expert datasets, SHARP-BC generally brings performance closer to the best attainable result across hyperparameter configurations, with the clearest consistency observed for ReBRAC. The cloned datasets require a more careful interpretation. On Hammer, Door, and Relocate, nearly all zero-shot and planning configurations achieve low absolute returns, consistent with prior observations that several D4RL Adroit datasets produce near-degenerate performance across offline RL methods (Jackson et al., 2025). In this regime, the experiment does not provide a meaningful test of hyperparameter robustness: a narrow distribution of uniformly poor returns reflects a failure of the underlying learned components, rather than successful robustness. SHARP-BC still depends on learned components provided by the offline RL and planning cannot recover when none of the evaluated configurations provides a sufficient information. Pen-cloned is the main exception: at least some configurations attain meaningful performance, and SHARP-BC both increases the median return and reduces variation across configurations. Overall, the Adroit results support the robustness benefit on the expert datasets and Pen-cloned, while also exposing an important limitation: SHARP-BC can stabilize usable underlying value functions but cannot rescue uniformly unsuccessful offline RL solutions.

E Baselines and Dataset Descriptions

E.1 Offline RL baselines

Here, we briefly introduce IQL and ReBRAC that serve as the underlying offline RL algorithms in our experiments. We select these methods following the systematic evaluation of Jackson et al. 2025, which finds that, while no single offline RL method consistently dominates across datasets, IQL and

ReBRAC are among the most consistently competitive model-free methods and identifies them as clear baselines for future method development.

Implicit Q-Learning (IQL; Kostrikov et al. 2022): IQL learns its critic only from offline dataset transitions and performs Bellman backups through a learned value function $V(s')$. Its actor is trained with advantage-weighted regression (AWR), which reweights dataset actions according to their estimated advantages (see Appendix F.1). The expectile and temperature parameters determine how strongly the resulting policy concentrates on higher-value behavior and therefore affect zero-shot performance.

Revisited BRAC (ReBRAC; Tarasov et al. 2023): ReBRAC builds on the behavior-regularized actor-critic (BRAC) framework. Unlike IQL, it forms critic targets using actions from the training policy while penalizing deviations from offline dataset actions in the critic objective. Its actor objective similarly combines standard Q-maximization with explicit behavioral regularization. These regularization coefficients control the trade-off between Q exploitation and behavioral regularization, directly affecting zero-shot performance. Achieving strong zero-shot performance, however, often requires dataset-specific tuning of these coefficients even within the same D4RL task, adding tuning overhead.

E.2 Plug-and-play planning methods

In this section, we discuss planning algorithms compatible with a plug-and-play planning paradigm, in which action proposers u and value functions $\hat{Q}$ learned via offline RL are incorporated into the planning process.

MBOP. Model-Based Offline Planning (MBOP; Argenson and Dulac-Arnold 2021), an early work extending model-based planning to the offline setting. It trains an ensemble of dynamic models $\{\hat{T}_e\}_{e=1}^{E}$, a behavior cloning policy as an action proposer u, and uses return-to-go as the objective to optimize the value functions $\hat{Q}$. For $a^k \sim u(\cdot \mid s_0)$, the trajectory is rolled out as:

$$a^k_{h+1} \sim u(\cdot \mid \hat{s}^k_{h+1}), \quad \hat{s}^k_{h+1} \sim \hat{T}_e(\cdot \mid \hat{s}^k_h, a^k_h) \quad e = k \bmod E \tag{12}$$

with $\hat{s}^k_0 = s_0, a^k_0 = a^k$. Then the return is estimated as:

$$G^k = \sum_{h=0}^{H-1} \gamma^h \hat{r}(\hat{s}^k_h, a^k_h) + \gamma^H \hat{Q}(\hat{s}^k_H, a^k_H), \tag{13}$$

Finally, the action to be executed in the environment is obtained by:

$$a^* = \frac{\sum_{k=1}^{K} \exp(\kappa G^k) a^k_0}{\sum_{k=1}^{K} \exp(\kappa G^k)}, \tag{14}$$

where κ is a hyperparameter controlling how strongly to emphasize actions with larger estimated return G^k. We consider its plug-and-play version in this work, where the action proposer u and value functions $\hat{Q}$ are from offline RL algorithms.

LOOP. Learning Off-Policy with Online Planning (LOOP; Sikchi et al. 2021) 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 $\hat{s}^k_{h,e,p}$, 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:

$$\begin{aligned} G^k_{e,p} &= \sum_{h=0}^{H-1} \gamma^h \hat{r}(\hat{s}^k_{h,e,p}, a^k_{h,e,p}) + \gamma^H \hat{Q}(\hat{s}^k_{H,e,p}, a^k_{H,e,p}), \\ G^k_e &= \text{mean}_p(G^k_{e,p}) - \beta_{pess} \cdot \text{std}_p(G^k_{e,p}), \quad G^k = \text{mean}_e(G^k_e), \end{aligned} \tag{15}$$

where β_{pess} controls the weight of the penalty. The action to be executed in the environment follows the same procedure as in MBOP.

Table 5: **Hyperparameters of model-based planning baselines.** We report the task-specific hyperparameters used by model-based planning methods. For MBOP, the implementation[1] used to obtain the reported results searches over $H \in \{4, 8, 16\}$, $K \in \{100, 200\}$, $\kappa \in \{0.01, 0.03, 0.1, 0.3\}$, and $\sigma \in \{0.05, 0.1, 0.2\}$, without specifying which configurations correspond to the reported scores. We therefore mark these entries as – in the table.

Environment	Data	MBOP				MOPP			
		H	κ	σ	K	H	κ	σ_M	K
HalfCheetah	M	2	3	0.2	100	2	3	0.45	100
	MR	4	3	0.2	100	4	3	0.50	100
	ME	2	1	0.2	100	2	1	0.55	100
Hopper	M	4	0.3	0.01	100	4	0.3	0.25	100
	MR	4	0.3	0.02	100	4	0.3	0.60	100
	ME	10	3	0.01	100	10	3	0.40	100
Walker2d	M	2	0.1	0.20	1000	2	0.1	0.55	1000
	MR	8	3	0.02	1000	8	3	0.20	1000
	ME	2	1	0.05	1000	2	1	0.40	1000
Pen	Cloned	–	–	–	–	4	0.30	0.8	100
	Expert	–	–	–	–	4	0.03	0.8	100
Hammer	Cloned	–	–	–	–	4	0.3	0.8	100
	Expert	–	–	–	–	4	0.3	0.7	100
Door	Cloned	–	–	–	–	4	0.30	0.8	100
	Expert	–	–	–	–	4	0.03	0.7	100
Relocate	Cloned	–	–	–	–	4	0.3	0.8	100
	Expert	–	–	–	–	16	0.3	0.4	100

[1] https://github.com/zhanzxy5/MOPP

Table 6: **Hyperparameters of sequence-based planning baselines.** We report the hyperparameters that yield the best performance for each method on D4RL. For TT, Wang et al. (2022) reports a sweep over $H \in \{5, 10, 15\}$, but does not specify the horizon used for the reported results. We therefore report this entry as – below.

Domain	TT		TAP				DV		
	H	B	H	B	L	β	H	N	M
Locomotion	15	256	15	64	3	0.05	32	50	1
Adroit	–	256	24	256	3	0.05	32	50	2

E.3 Offline planning baselines

To provide a comprehensive comparison, we select methodologies that represent two paradigms in offline reinforcement learning and planning: sequence-based and model-based approaches. We briefly summarize the core mechanics of each baseline and report previously published scores. The reported scores in Table 2 are taken from Tables 1 and 2 of zhengyao jiang et al. (2023) for TT and TAP, Tables 8 and 12 of Lu et al. (2025) for DV, and Tables 1 and 2 of Zhan et al. (2022) for MBOP and MOPP. Sequence-based planning methods reformulate the reinforcement learning and planning problem as a sequence modeling or generative task, where the latter typically leverages generative models to co-learn the planning process and the dynamic model.

Trajectory Transformer (TT; Janner et al. 2021): Trajectory Transformer (TT) represents offline trajectories as sequences of discretized state, action, and reward tokens and trains a Transformer to autoregressively predict future trajectory tokens. Given the current state, TT searches over candidate future trajectories using beam search (with beam width B) and selects the one with the highest return for receding-horizon control. It still requires tuning the planning horizon H.

Trajectory Autoencoding Planner (TAP; zhengyao jiang et al. 2023): Building on TT, Trajectory Autoencoding Planner (TAP) performs planning in a discrete latent action space to reduce the difficulty of searching directly in high-dimensional action spaces. During training, an encoder compresses offline trajectories into discrete latent codes, while a Transformer autoregressively models

the distribution over these latent-code sequences. At inference time, TAP searches over the latent space for codes corresponding to high-return trajectories and executes the first action of the selected trajectory. TAP introduces additional hyperparameters beyond the planning horizon, including the latent temporal abstraction L and probability threshold β.

Diffusion Veteran (DV; Lu et al. 2025): Diffusion Veteran takes a fundamentally different generative approach by formulating planning as a denoising diffusion process. Rather than explicitly rolling out a learned dynamics model, DV employs a diffusion transformer to generate candidate state plans. At test time, it generates N candidate plans with planning horizon H and jump-step length M, where each plan consists of the state sequence $(s_t, s_{t+M}, \ldots, s_{t+(H-1)M})$. The candidate with the highest predicted return is selected using the critic, and an inverse dynamics model is then used to recover the corresponding actions. In addition to the planning horizon, DV therefore requires tuning the planning stride M.

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

MBOP (Argenson and Dulac-Arnold, 2021) is discussed in Appendix E.2. During planning, it rolls out K candidate actions; each is perturbed by a small noise based on σ, uses the dynamics models to evaluate their returns at horizon H, and selects an action by softmax weighting the candidates with κ. We use its original version for the performance in Table 2.

Model-Based Offline Planning with Trajectory Pruning (MOPP; Zhan et al. 2022): Model-Based Offline Planning with Trajectory Pruning (MOPP) addresses the over-conservatism of behavior-guided offline planning by allowing more aggressive action sampling around the learned behavior policy, while pruning uncertain model rollouts to avoid unreliable OOD trajectories. It further uses offline Q-functions to bias sampled actions toward higher-return regions. In addition to the planning horizon, MOPP introduces parameters such as the action-sampling std scaling σ_M.

In Table 5 and 6, we summarize the hyperparameters used by each baseline. Overall, existing methods typically rely on task-specific tuning or introduce additional planning-specific hyperparameters, increasing the tuning overhead across tasks and datasets. In contrast, for SHARP-BC, we simply use the largest values considered in our experiments, setting $H = 8$ and $\kappa = 10$ across all tasks in Table 2, further highlighting the robustness of our method and its reduced sensitivity to hyperparameter choices.

E.4 Tasks and datasets

We evaluate our methods on seven continuous-control benchmark tasks from D4RL (Fu et al., 2020): three standard MuJoCo locomotion environments (*halfcheetah*, *hopper*, and *walker2d*) and four Adroit dexterous manipulation tasks (*pen*, *hammer*, *door*, and *relocate*). These domains cover two fundamentally distinct challenges in model-based planning: long-horizon locomotion and high-dimensional contact-rich manipulation. While SHARP focuses on the locomotion benchmarks, SHARP-BC is evaluated across both domains to examine its broader applicability.

For each locomotion environment, we evaluate across five standard dataset configurations: **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 up to medium performance; **medium-expert** (ME), mixing medium-level and expert trajectories; and **full-replay** (FR), containing the entire training replay buffer. In total, these 15 combinations span wide variations in behavioral coverage and distribution shift.

For the manipulation domain, each task includes two dataset types: **expert** (E), collected via an expert policy, and **cloned** (C), which augments the human demonstrations with rollouts from a behavior-cloned policy. Across both domains, the resulting 23 task–dataset pairs provide a comprehensive benchmark for evaluating planning robustness under diverse action dimensionalities and data regimes.

Table 7: **Offline RL training hyperparameters.** We list the IQL, ReBRAC, and policy extraction hyperparameters used in all experiments.

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]
ReBRAC	Steps	1M
	Batch size	256 (1024 on Gym-MuJoCo)
	Target update rate	5e-3
	Actor lr	3e-4 (1e-3 on Gym-MuJoCo)
	Critic lr	3e-4 (1e-3 on Gym-MuJoCo)
	Actor hidden dim	[256, 256, 256]
	Critic hidden dim	[256, 256, 256]
	Gamma	0.99
BC/AWR/DDPG+BC	Steps	1M
	Batch size	256
	Learning rate	3e-4
	Actor hidden dim	[256, 256]
	Policy type	Stochastic

F Implementation details

F.1 BC and Offline RL

Offline RL. We use IQL (Kostrikov et al., 2022) and ReBRAC (Tarasov et al., 2023) as the underlying offline RL algorithms. We follow the implementations and hyperparameter configurations from Jax-CORL (Nishimori, 2024) and the original ReBRAC implementation (Tarasov et al., 2023), respectively. The corresponding training hyperparameters are summarized in Table 7.

Action proposer training. For the policy-extraction experiments in Section 4, we compare behavior cloning (BC) with two standard policy-extraction objectives that incorporate Q-function exploitation: advantage-weighted regression (AWR) (Peng et al., 2019) and DDPG+BC (Fujimoto and Gu, 2021). All three policies use the same network architecture and training budget to isolate the effect of the policy-extraction objective.

(1) Advantage-Weighted Regression (AWR) (Peng et al., 2019). AWR reweights behavior actions according to their estimated advantages:

$$\max_{\pi} \; \mathcal{J}_{\text{AWR}}(\pi) = \mathbb{E}_{s,a\sim\mathcal{D}}\left[e^{\alpha(Q(s,a)-V(s))}\log\pi(a \mid s)\right], \tag{16}$$

where α controls the strength of Q-function exploitation. When $\alpha = 0$, all dataset actions receive equal weight and the objective reduces to standard behavioral cloning; increasing α places greater weight on actions with higher estimated advantage.

(2) Deep Deterministic Policy Gradient (DDPG) (Fujimoto and Gu, 2021). We additionally consider a behavior-regularized policy-gradient objective:

$$\max_{\pi} \; \mathcal{J}_{\text{DDPG+BC}}(\pi) = \mathbb{E}_{s,a\sim\mathcal{D}}\left[Q(s,\mu^{\pi}(s)) + \alpha\log\pi(a \mid s)\right], \tag{17}$$

where $\mu_\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[a]$. Here, α controls the strength of behavioral regularization: smaller values place greater emphasis on maximizing the learned Q-function, whereas larger values place greater emphasis on matching the behavior data.

(3) Behavior Cloning (BC). BC policy is trained without using the learned Q-function by maximizing the log-likelihood of the dataset actions:

$$\max_{\pi} J_{\mathrm{BC}}(\pi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\log \pi(a \mid s)\right]. \tag{18}$$

We select the policy output parameterization entirely offline according to the BC training loss, without environment evaluation. Specifically, we apply a $\tanh$ output transformation for the locomotion tasks, whereas we use an unsquashed output for the Adroit tasks, as these choices yield lower offline training MSE in their respective domains.

For zero-shot policy evaluation, we execute the deterministic policy prediction. When a policy is used as an action proposer for planning, we instead sample from its stochastic policy to provide diverse candidate actions. The training hyperparameters for BC, AWR, and DDPG+BC are summarized in Table 7.

F.2 Plug-and-play planning

We train the dynamics model following the setting of previous work Sikchi et al. (2021), and the hyperparameters are provided in Table 8.

Table 8: **Hyperparameters for learned dynamics and reward models.** We use the same network architecture and training configuration across all 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

For all the plug-and-play planning methods, we set K=32 and a small σ=0.01 as in Equation 7 to prevent all the candidate actions from being identical. For LOOP and SHARP, we set particle P=4. We set β_{pess}=0.1 for LOOP.

G Experiments compute resources

We further analyze the computational cost of SHARP compared with other planning methods using IQL as the action proposer. As shown in Table 9, despite performing ensemble rollouts, SHARP maintains a runtime comparable to that of other planning methods across all planning horizons by efficiently batching rollout and value computations.

Table 9: **Planning time per environment step (ms).** Results are averaged over 3 seeds and 5 evaluation episodes and reported for different planning methods and planning horizons H across the locomotion and manipulation tasks. All the results are estimated on an NVIDIA H100 GPU.

Environment	**Method**	**Planning Horizon H**							
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
HalfCheetah	MBOP	6	8	10	13	15	17	19	21
	LOOP	9	12	15	18	22	25	28	31
	SHARP (Ours)	7	10	12	15	17	19	22	24
Hopper	MBOP	8	10	12	14	16	18	20	22
	LOOP	11	15	18	21	24	28	30	33
	SHARP (Ours)	9	11	14	16	19	21	23	26
Walker2d	MBOP	6	8	11	13	15	17	19	21
	LOOP	9	12	15	19	22	25	28	32
	SHARP (Ours)	7	10	12	15	17	20	22	24
Pen	MBOP	17	20	22	24	26	28	30	32
	LOOP	24	28	31	34	37	41	44	47
	SHARP (Ours)	19	22	25	28	30	32	35	37
Hammer	MBOP	12	15	17	19	21	23	25	27
	LOOP	17	20	24	27	30	33	36	40
	SHARP (Ours)	14	16	19	21	24	26	28	31
Door	MBOP	12	14	16	19	20	23	24	27
	LOOP	17	20	23	26	30	33	36	39
	SHARP (Ours)	13	16	19	21	23	26	28	31
Relocate	MBOP	12	15	17	19	21	23	25	28
	LOOP	17	21	24	27	30	33	36	40
	SHARP (Ours)	14	17	19	22	24	27	29	32