arXiv:2008.10066v5 [cs.LG] 5 Oct 2021

Learning Off-Policy with Online Planning

Harshit Sikchi,* **Wenxuan Zhou**, **David Held**
Robotics Institute
Carnegie Mellon University
{hsikchi, wenxuanz, dheld}@cs.cmu.edu

Abstract: Reinforcement learning (RL) in low-data and risk-sensitive domains requires performant and flexible deployment policies that can readily incorporate constraints during deployment. One such class of policies are the semi-parametric H-step lookahead policies, which select actions using trajectory optimization over a dynamics model for a fixed horizon with a terminal value function. In this work, we investigate a novel instantiation of H-step lookahead with a *learned* model and a terminal value function learned by a *model-free off-policy* algorithm, named Learning Off-Policy with Online Planning (LOOP). We provide a theoretical analysis of this method, suggesting a tradeoff between model errors and value function errors and empirically demonstrate this tradeoff to be beneficial in deep reinforcement learning. Furthermore, we identify the "Actor Divergence" issue in this framework and propose Actor Regularized Control (ARC), a modified trajectory optimization procedure. We evaluate our method on a set of robotic tasks for Offline and Online RL and demonstrate improved performance. We also show the flexibility of LOOP to incorporate safety constraints during deployment with a set of navigation environments. We demonstrate that LOOP is a desirable framework for robotics applications based on its strong performance in various important RL settings. Project video and details can be found at hari-sikchi.github.io/loop.

Keywords: Reinforcement Learning, Trajectory Optimization, Safety

1 Introduction

Off-policy reinforcement learning algorithms have been widely used in many robotic applications due to their sample efficiency and their ability to incorporate data from different sources [1, 2, 3, 4]. Model-free off-policy algorithms sample transitions from a replay buffer to learn a value function and then update the policy according to the value function [5, 6]. Thus, the performance of the policy is highly dependent on the estimation of the value function. However, learning an accurate value function from off-policy data is challenging especially in deep RL due to a variety of issues, such as overestimation bias [7, 8], delusional bias [9], rank loss [10], instability [11], and divergence [12]. Another shortfall of model-free off-policy algorithms in continuous control is that the policy is usually parametrized by a feedforward neural network which lacks flexibility during deployment.

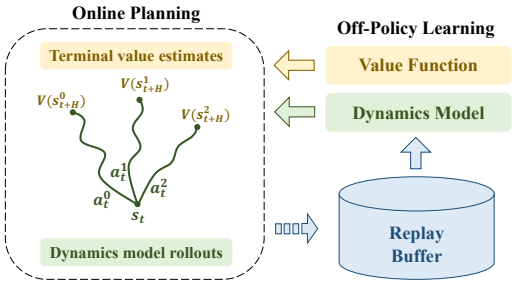

Figure 1: Overview of LOOP: A learned dynamics model is utilized for Online Planning with a terminal value function. The value function is learned via a model-free off-policy algorithm.

Previous works in model-based RL have explored different ways of using a dynamics model to improve off-policy algorithms [13, 14, 15, 16, 17]. One way of incorporating the dynamics model is to use H-step lookahead policies [18]. At each timestep, H-step lookahead policies rollout the dynamics model H-step into the future from the current state to find an action sequence with the highest return. Within this trajectory optimization process, a terminal value function is attached to the end of the rollouts to provide an estimation of the return beyond the fixed horizon. This way of online

*Currently at The University of Texas at Austin, Email: hsikchi@utexas.edu

5th Conference on Robot Learning (CoRL 2021), London, UK.

planning offers us a degree of explainability missing in fully parametric methods while also allowing us to take constraints into account during deployment. Previous work proves faster convergence with H-step lookahead policies in tabular setting [18] or showed improved sample complexity with a *ground-truth* dynamics model [19]. However, the benefit of H-step lookahead policies remains unclear under an *approximate model* and an *approximate value function*. Additionally, if H-step lookahead policies are used during the value function update [19], the required computation of value function update will be significantly increased.

In this work, we take this direction further by studying H-step lookahead both theoretically and empirically with three main contributions. **First**, we provide a theoretical analysis of H-step lookahead under an *approximate model* and *approximate value function*. Our analysis suggests a trade-off between model error and value function error, and we empirically show that this tradeoff can be used to improve policy performance in Deep RL. **Second**, we introduce Learning Off-Policy with Online Planning (LOOP) (Figure 1). To avoid the computational overhead of performing trajectory optimization while updating the value function as in previous work [19], the value function of LOOP is updated via a parameterized actor using a model-free off-policy algorithm ("Learning Off-Policy"). LOOP exploits the benefits of H-step lookahead policies when the agent is deployed in the environment during exploration and evaluation ("Online Planning"). This novel combination of model-based online planning and model-free off-policy learning provides sample-efficient and computationally-efficient learning. We also identify the "Actor Divergence" issue in this combination and propose a modified trajectory optimization method called Actor Regularized Control (ARC). ARC performs implicit divergence regularization with the parameterized actor through Iterative Importance Sampling.

Third, we explore the flexibility of H-step lookahead policies for improved performance in offline RL and safe RL, which are both important settings in robotics. LOOP can be applied on top of various offline RL algorithms to improve their evaluation performance. LOOP's semiparameteric behavior policy also allows it to easily incorporate safety constraints during deployment. We evaluate LOOP on a set of simulated robotic tasks including locomotion, manipulation, and controlling an RC car. We show that LOOP provides significant improvement in performance for online RL, offline RL, and safe RL, which makes it a strong choice of RL algorithm for robotic applications.

2 Related Work

Model-based RL Model-based reinforcement learning (MBRL) methods learn a dynamics model and use it to optimize the policy. State-of-the-art model-based RL methods usually have better sample efficiency compared to model-free methods while maintaining competitive asymptotic performance [20, 13]. One approach in MBRL is to use trajectory optimization with a learned dynamics model [17, 21, 22]. These methods can reach optimal performance when a large enough planning horizon is used. However, they are limited by not being able to reason about the rewards beyond the planning horizon. Increasing the planning horizon increases the number of trajectories that need to be sampled and incurs a heavy computational cost.

Various attempts have been made to combine model-free and model-based RL. GPS [23] combines trajectory optimization using analytical models with the on-policy policy gradient estimator. MBVE [15] and STEVE [16] use the model to improve target value estimates. Approaches such as MBPO [13] and MAAC [24] follow Dyna-style [25] learning where imagined short-horizon trajectories are used to provide additional transitions to the replay buffer leveraging model generalization. Piché et al. [26] use Sequential Monte Carlo (SMC) to capture multimodal policies. The SMC policy relies on combining multiple 1-step lookahead value functions to sample a trajectory proportional to the unnormalized probability $\exp(\sum_{i=1}^{H}(A(s,a)))$; this approach potentially compounds value function errors, in contrast to LOOP which uses single H-step lookahead planning for each state. POLO [19] shows advantages of trajectory optimization under *ground-truth* dynamics with a terminal value function. The value function updates involve additional trajectory optimization routines which is one of the issues we aim to address with LOOP. The computation of trajectory optimization in POLO is $\mathcal{O}(THN)$ while LOOP is $\mathcal{O}(TH)$ where T is the number of environment timesteps, H is the planning horizon, and N is the number of samples needed for training the value function.

Off-Policy RL LOOP relies on a terminal value function for long horizon reasoning which can be learned effectively via model-free off-policy RL algorithms. Off-policy RL methods such as SAC [5]

and TD3 [6] use the replay buffer to learn a Q-function that evaluates a parameterized actor and then optimize the actor by maximizing the Q-function. Off-policy methods can be modified to be used for Offline RL problems where the goal is to learn a policy from a static dataset [27, 8, 28, 29, 30, 31, 32]. MBOP [33], a recent model-based offline RL method, leverages planning with a terminal value function, but the value function is a Monte Carlo evaluation of truncated replay buffer trajectories, whereas in LOOP the value function is trained for optimality under the dataset.

3 Preliminaries

A Markov Decision Process (MDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \rho_0)$ with state-space $\mathcal{S}$, action-space $\mathcal{A}$, transition probability $p(s_{t+1}|s_t, a_t)$, reward function $r(s, a)$, and initial state distribution $\rho_0(s)$. In the infinite horizon discounted MDP, the goal of reinforcement learning algorithms is to maximize the return for policy π given by $J^\pi = \mathbb{E}_{a_t \sim \pi(s_t), s_0 \sim \rho_0}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$.

Value functions: $V^\pi : \mathcal{S} \to \mathbb{R}$ represents a state-value function which estimates the return from the current state s_t and following policy π, defined as $V^\pi(s) = \mathbb{E}_{a_t \sim \pi(s_t)}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 = s]$. Similarly, $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ represents a action-value function, usually referred as a Q-function, defined as $Q^\pi(s, a) = \mathbb{E}_{a_t \sim \pi(s_t)}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 = s, a_0 = a]$. Value functions corresponding to the optimal policy π^* are defined to be V^* and Q^*. The value function can be updated according to the Bellman operator $\mathcal{T}$:

$$\mathcal{T}Q(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim p, a_{t+1} \sim \pi_Q}[\gamma(Q(s_{t+1}, a_{t+1})] \tag{1}$$

where π_Q is updated to be greedy with respect to Q, the current Q-function.

Constrained MDP for safety: A constrained MDP (CMDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r, c, \rho_0)$ with an additional cost function $c(s, a)$. We define the cumulative cost of a policy to be $D^\pi = \mathbb{E}_{a_t \sim \pi(s_t), s_0 \sim \rho_0}[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t)]$. A common objective for safe reinforcement learning is to find a policy $\pi = \text{argmax}_\pi J^\pi$ subject to $D^\pi \leq d_0$ where d_0 is a safety threshold [34].

4 H-step Lookahead with Learned Model and Value Function

Model-based algorithms often learn an approximate dynamics model $\hat{M}(s_{t+1}|s_t, a_t)$ using the data collected from the environment. One way of using the model is to find an action sequence that maximizes the cumulative reward with the learned model using trajectory optimization [35, 36, 37]. An important limitation of this approach is that the computation grows exponentially with the planning horizon. Thus, methods like [35, 17, 21, 38, 39] plan over a fixed, short horizon and are unable to reason about long-term reward. Let π_H be such a fixed horizon policy:

$$\pi_H(s_0) = \operatorname*{argmax}_{a_0} \max_{a_1, .., a_{H-1}} \mathbb{E}_{\hat{M}}[R_H(s_0, \tau)] \text{ , where } R_H(s_0, \tau) = \sum_{t=0}^{H-1} \gamma^t r(s_t, a_t) \tag{2}$$

where τ denotes the action sequence $a_{[0..H-1]}$. One way to enable efficient long-horizon reasoning is to augment the planning trajectory with a terminal value function. Given a value-function $\hat{V}$, we define a policy $\pi_{H,\hat{V}}$ obtained by maximizing the H-step lookahead objective:

$$\pi_{H,\hat{V}}(s_0) = \operatorname*{argmax}_{a_0} \max_{a_1, .., a_{H-1}} \mathbb{E}_{\hat{M}}\left[R_{H,\hat{V}}(s_0, \tau)\right] \tag{3}$$

$$\text{where } R_{H,\hat{V}}(s_0, \tau) = \sum_{t=0}^{H-1} \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)$$

The quality of both the model $\hat{M}$ and the value-function $\hat{V}$ affects the performance of the overall policy. To show the benefits of this combination of model-based trajectory optimization and the value-function, we now analyze and bound the performance of the H-step look-ahead policy $\pi_{H,\hat{V}}$ compared to its fixed-horizon counterpart without the value-function π_H (Eqn. 2), as well as the greedy policy obtained from the value-function $\pi_{\hat{V}} = \text{argmax}_a \mathbb{E}_{s' \sim M(.|s,a)}\left[r(s, a) + \gamma \hat{V}(s')\right]$. Following previous work, we will construct the proofs with the state-value function V, but the proofs for the action-value function Q can be derived similarly.

Lemma 1. *(Singh and Yee [40]) Suppose we have an approximate value function $\hat{V}$ such that $\max_s |V^*(s) - \hat{V}(s)| \leq \epsilon_v$. Then the performance of the 1-step greedy policy $\pi_{\hat{V}}$ can be bounded as:*

$$J^{\pi^*} - J^{\pi_{\hat{V}}} \leq \frac{\gamma}{1-\gamma}[2\epsilon_v] \tag{4}$$

Theorem 1. *(H-step lookahead policy) Suppose $\hat{M}$ is an approximate dynamics model with Total Variation distance bounded by ϵ_m. Let $\hat{V}$ be an approximate value function such that $\max_s |V^*(s) - \hat{V}(s)| \leq \epsilon_v$. Let the reward function $r(s,a)$ be bounded by [0,R_{max}] and $\hat{V}$ be bounded by [0,V_{max}]. Let ϵ_p be the suboptimality incurred in H-step lookahead optimization (Eqn. 3). Then the performance of the H-step lookahead policy $\pi_{H,\hat{V}}$ can be bounded as:*

$$J^{\pi^*} - J^{\pi_{H,\hat{V}}} \leq \frac{2}{1-\gamma^H}[C(\epsilon_m, H, \gamma) + \frac{\epsilon_p}{2} + \gamma^H \epsilon_v] \tag{5}$$

where

$$C(\epsilon_m, H, \gamma) = R_{\max} \sum_{t=0}^{H-1} \gamma^t t \epsilon_m + \gamma^H H \epsilon_m V_{max}$$

Proof. Due to the page limit, we defer the proof to Appendix A.1. We also provide extension of Theorem 1 under assumptions on model generalization and concentrability in Corollary 1 and Theorem 2 respectively in Appendix A. □

H-step Lookahead Policy vs H-step Fixed Horizon Policy: The fixed-horizon policy π_H can be considered as a special case of $\pi_{H,\hat{V}}$ with $\hat{V}(s) = 0 \; \forall s \in \mathcal{S}$. Following Theorem 1, $\epsilon_{\hat{V}} = \max_s |V^*(s)|$ implies a potentially large optimality gap. This suggests that learning a value function that better approximates V^* than $\hat{V}(s) = 0$ will give us a smaller optimality gap in the worst case.

H-step lookahead policy vs 1-step greedy policy: By comparing Lemma 1 and Theorem 1, we observe that the performance of the H-step lookahead policy $\pi_{H,\hat{V}}$ reduces the dependency on the value function error ϵ_v at least by a factor of γ^{H-1} while introducing an additional dependency on the model error ϵ_m. This implies that the H-step lookahead is beneficial when the value-function bias dominates the bias in the learned model. In the low data regime, the value function bias can result from compounded sampling errors [41] and is likely to dominate the model bias, as evidenced by the success of model-based RL methods in the low-data regime [33, 42, 13]; we observe this hypothesis to be consistent with our experiments where H-step lookahead offers large gains in sample efficiency. Further, errors in value learning with function approximation can stem from a number of reasons explored in previous work, some of them being Overestimation, Rank Loss, Divergence, Delusional bias, and Instability [7, 11, 6, 43, 10]. Although this result may be intuitive to many practitioners, it has not been shown theoretically; further, we demonstrate that we can use this insight to improve the performance of state-of-the-art methods for online RL, offline RL, and safe RL.

5 Learning Off-Policy with Online Planning

We propose Learning Off-Policy with Online Planning (LOOP) as a framework of using H-step lookahead policies that combines online trajectory optimization with model-free off-policy RL (Figure 1). We use the replay buffer to learn a dynamics model and a value function using an off-policy algorithm. The H-step lookahead policy (Eqn. 3) generates rollouts using the dynamics model with a terminal value function and selects the best action for execution. The underlying off-policy algorithm is boosted by the H-step lookahead which improves the performance of the policy during both exploration and evaluation. From another perspective, the underlying model-based trajectory optimization is improved using a terminal value function for reasoning about future returns. In this section, we discuss the Actor Divergence issue in the LOOP framework and introduce additional applications and instantiations of LOOP for offline RL and safe RL.

5.1 Reducing actor-divergence with Actor Regularized Control (ARC)

As discussed above, LOOP utilizes model-free off-policy algorithms to learn a value function in a more computationally efficient manner. It relies on actor-critic methods which use a parametrized actor π_θ to facilitate the Bellman backup. However, we observe that combining trajectory optimization and policy learning naively will lead to an issue that we refer to as "actor divergence": a different policy is used for data collection (H-step lookahead policy $\pi_{H,\hat{V}}$) than the policy that is used to learn

the value-function (the parametrized actor π_θ). This leads to a potential distribution shift between the state-action visitation distribution between the parametrized actor π_θ and the actual behavior policy $\pi_{H,\hat{V}}$ which can lead to accumulated bootstrapping errors with the Bellman update and destabilize value learning [43]. One possible solution in this case is to use Offline RL [30]; however, in practice, we observe that offline RL in this setup leads to learning instabilities. We defer discussion on this alternative to the Appendix D.7. Instead, we propose to resolve the actor-divergence issue via a modified trajectory optimization method called Actor Regularized Control (ARC).

In ARC, we aim to constrain the action selection of the trajectory optimization to be close to the parametrized actor. We frame the following general constrained optimization problem for policy improvement [44]:

$$p^\tau_{opt} = \underset{p^\tau}{\operatorname{argmax}}\, \mathbb{E}_{p^\tau}[L_{H,\hat{V}}(s_t, \tau)]\ ,\ \text{s.t}\ \ D_{KL}(p^\tau || p^\tau_{prior}) \leq \epsilon \tag{6}$$

where $L_{H,\hat{V}}(s_t, \tau)$ is the expected lookahead objective (Eqn. 3) under the learned model given by $L_{H,\hat{V}}(s_t, \tau) = \mathbb{E}_{\hat{M}}\Big[R_{H,\hat{V}}(s_t, \tau)\Big]$, starting from state s_t, p^τ is a distribution over action sequences τ of horizon H starting from s_t, and p^τ_{prior} is a prior distribution over such action sequences. We will use the parametrized actor to derive this prior in ARC. This optimization admits a closed form solution by enforcing the KKT conditions where the optimal policy is given by $p^\tau_{opt} \propto p^\tau_{prior} e^{\frac{1}{\eta} L_{H,\hat{V}}(s_t,\tau)}$ [45, 46, 47, 48], where η is the lagrangian dual variable. The above formulation generalizes a number of prior work [5, 35, 45] (more details in Appendix B.3).

Approximating the optimal policy p^τ_{opt} as a multivariate gaussian with diagonal covariance $\hat{p}^\tau_{opt} = \mathcal{N}(\mu_{opt}, \sigma_{opt})$, the parameters can be estimated using importance sampling under the proposal distribution p^τ_{prior} as:

$$\hat{p}^\tau_{opt} = \mathcal{N}(\mu_{opt}, \sigma_{opt})\ ,\ \mu_{opt} = \mathbb{E}_{\tau', \hat{M}}\left[\frac{p^\tau_{opt}(\tau')}{p^\tau_{prior}(\tau')}\tau'\right]\ ,\ \sigma_{opt} = \mathbb{E}_{\tau', \hat{M}}\left[\frac{p^\tau_{opt}(\tau')}{p^\tau_{prior}(\tau')}(\tau' - \mu)^2\right] \tag{7}$$

where $\tau' \sim p^\tau_{prior}$. We use iterative importance sampling to estimate $\hat{p}^\tau_{opt}$ which is parameterized as a Gaussian whose mean and variance at iteration $m+1$ are given by the empirical estimate:

$$\mu^{m+1} = \frac{\sum_{i=1}^N [e^{\frac{1}{\eta} L_{H,\hat{V}}(s_t,\tau')}\tau']}{\sum_{i=1}^N e^{\frac{1}{\eta} L_{H,\hat{V}}(s_t,\tau')}}\ ,\ \sigma^{m+1} = \frac{\sum_{i=1}^N [e^{\frac{1}{\eta} L_{H,\hat{V}}(s_t,\tau')}(\tau' - \mu^{m+1})^2]}{\sum_{i=1}^N e^{\frac{1}{\eta} L_{H,\hat{V}}(s_t,\tau')}} \tag{8}$$

where $\tau' \sim \mathcal{N}(\mu^m, \sigma^m)$ and $\mathcal{N}(\mu^0, \sigma^0)$ is set to p^τ_{prior}. As long as we perform a finite number of iterations, the final trajectory distribution is constrained in total variation to be close to the prior as a result of finite trust region updates as shown in Lemma 2 in Appendix A.4.

To reduce actor divergence in LOOP, we constrain the action-distribution of the trajectory optimization to be close to that of the parametrized actor π_θ. To do so, we set $p^\tau_{prior} = \beta\pi_\theta + (1-\beta)\mathcal{N}(\mu_{t-1}, \sigma)$. The trajectory prior is a mixture of the parametrized actor and the action sequence from the previous environment timestep with additional Gaussian noise $\mathcal{N}(0, \sigma)$. Using 1-timestep shifted solution from the previous timestep allows to amortize trajectory optimization over time [33]. For online RL, we can vary σ to vary the amount of exploration during training. For offline RL, we set $\beta = 1$ to constrain actions to be close to those in the dataset (from which π_θ is learned) to be more conservative.

5.2 Additional instantiations of LOOP: Offline-LOOP and Safe-LOOP

LOOP not only improves the performance of previous model-based and model-free RL algorithms but also shows versatility in different settings such as the offline RL setting and the safe RL setting. These potentials of H-step lookahead policies have not been explored in previous work.

LOOP for Offline RL: In offline reinforcement learning, the policy is learned from a static dataset without further data collection. We can use LOOP on top of an existing off-policy algorithm as a plug-in component to improve its test time performance by using the model-based rollouts as suggested by Theorem 1. Note that this is different from the online setting in the previous section in which LOOP also influences exploration. In offline-LOOP, to account for the uncertainty in the model and the Q-function, ARC optimizes for the following uncertainty-pessimistic objective similar to [49, 50]:

$$\text{mean}_{[K]}[R_{H,\hat{V}}(s_t, \tau)] - \beta_{pess}\text{std}_{[K]}[R_{H,\hat{V}}(s_t, \tau)] \tag{9}$$

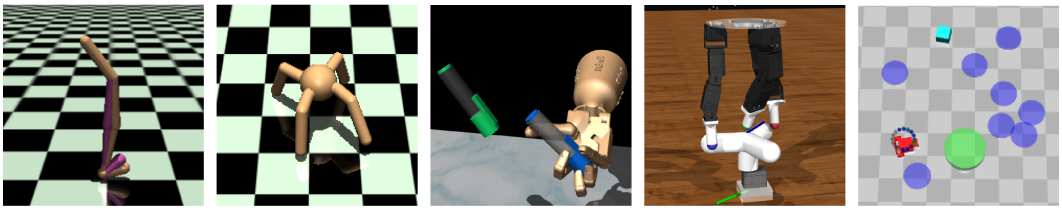

Figure 2: We evaluate LOOP over a variety of environments ranging from locomotion, manipulation to navigation including Walker2d-v2, Ant-v2, PenGoal-v1, Claw-v1, CarGoal1, etc.

where $[K]$ are the model ensembles, β_{pess} is the pessimism parameter and $R_{H,\hat{V}}$ is the H-horizon lookahead objective defined in Eqn. 3.

Safe Reinforcement Learning: Another benefit of LOOP with its semi-parameteric policy is that we can easily incorporate (possibly non-stationary) constraints with the model-based rollout, while being an order of magnitude more sample efficient than existing safe model-free RL algorithms. To account for safety in the planning horizon, ARC optimizes for the following cost-pessimistic objective:

$$\text{argmax}_{a_t} \mathbb{E}_{\hat{M}}\Big[R_{H,\hat{V}}(s_t,\tau)\Big] \text{s.t. } \max_{[K]} \sum_{t=t}^{t+H-1} \gamma^t c(s_t,a_t) \leq d_0 \tag{10}$$

where $[K]$ are the model ensembles, c is the constraint cost function and $R_{H,\hat{V}}$ is the H-horizon lookahead objective defined in Eqn. 3 and d_0 is the constraint threshold. For each action rollout, the worst-case cost is considered w.r.t model uncertainty to be more conservative. The pseudocode for modified ARC to solve the above constrained optimization is given in Appendix B.3.1.

6 Experimental Results

In the experiments, we evaluate the performance of LOOP combined with different off-policy algorithms in the settings of online RL, offline RL and safe RL over a variety of environments (Figure 2). Implementation details of LOOP and the baselines can be found in Appendix C.

6.1 LOOP for Online RL

In this section, we evaluate the performance of LOOP for online RL on three OpenAI Gym MuJoCo [51] locomotion control tasks: `HalfCheetah-v2`, `Walker-v2`, `Ant-v2` and two manipulation tasks: `PenGoal-v1`, `Claw-v1`. In these experiments, we use Soft Actor-Critic (SAC) [5] as the underlying off-policy method with the ARC optimizer described in Section 5.1. Further experiments on `InvertedPendulum-v2`, `Swimmer`, `Hopper-v2` and `Humanoid-v2` and more details on the baselines can be found in Appendix D.1 and Appendix C.2 respectively.

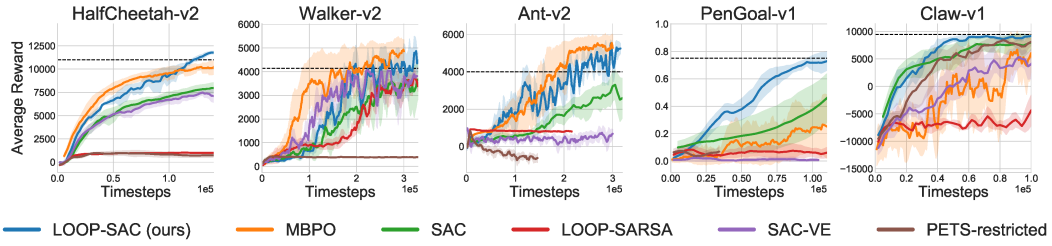

Figure 3: Comparisons of LOOP and the baselines for online RL. LOOP-SAC is significantly more sample efficient than SAC. It is competitive to MBPO for locomotion tasks and outperforms MBPO for manipulation tasks (PenGoal-v1 and Claw-v1). The dashed line indicates the performance of SAC at 1 million timesteps. Additional results on more environments can be found in Appendix D.1.

Baselines: We compare the LOOP framework against the following baselines: PETS-restricted, a variant of PETS [17] that uses trajectory optimization (CEM) for the same horizon as LOOP but without a terminal value function. LOOP-SARSA uses a terminal value function which is an evaluation of the replay buffer policy, similar to MBOP [33] in spirit. To compare with other ways of combining model-based and model-free RL, we also compare against MBPO [13] and SAC-VE. MBPO leverages the learned model to generate additional transitions for value function learning.

SAC-VE utilizes the model for value expansion, similar to MBVE [15] but uses SAC as the model-free component for a fair comparison with LOOP as done in [13]. We do not include comparison to STEVE [16] or SLBO [52] as they were shown to be outperformed by MBPO, and perform poorly compared to SAC in Hopper and Walker environments [13]. We were unable to reproduce the results for SMC [26] due to missing implementation. We did not include POLO here due several reasons. An extended discussion can be found in Appendix D.2.

Performance: From Figure 3, we observe that LOOP-SAC is significantly more sample efficient than SAC, the underlying model-free method used to learn a terminal value function. LOOP-SAC also scales well to high-dimensional environments like `Ant-v2` and `PenGoal-v1`. PETS-restricted performs poorly due to myopic reasoning over a limited horizon H. SAC-VE and MBPO represent different ways of incorporating a model to improve off-policy learning. LOOP-SAC outperforms SAC-VE and performs competitively to MBPO, outperforming it significantly in `PenGoal-v1` and `Claw-v1`. In principle, methods like MBPO and value expansion can be combined with LOOP to potentially increase performance; we leave such combinations for future work. LOOP-SARSA has poor performance as a result of the poor value function that is trained for evaluating replay buffer policy rather than optimality. As an ablation study, we also run experiments using LOOP without ARC, which optimizes the unconstrained objective of Eqn. 3 using CEM [36]. Figure 4 (left) shows that ARC reduces actor-divergence effectively and Figure 4 (right) shows that learning performance is poor in absence of ARC for `Walker-v2`. More ablation results can be found in Appendix D.5.

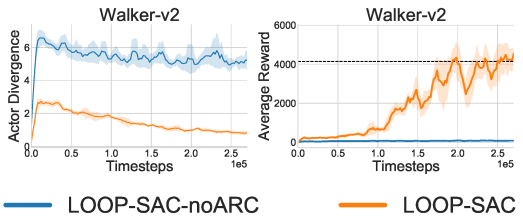

Figure 4: (Left) ARC reduces the actor-divergence measured by the L2 distance between the mean of the parametrized actor and the output of the H-step lookahead policy. (Right) In absence of ARC, policy learning can be unstable.

6.2 LOOP for Offline RL

Dataset	Env	CRR	LOOP CRR	Improve%	PLAS	LOOP PLAS	Improve%	MBOP
medium	hopper	65.73	**85.83**	30.6	32.08	56.47	76.0	48.8
	halfcheetah	41.14	41.54	1.0	39.33	39.54	0.5	**44.6**
	walker2d	69.98	**79.18**	13.1	46.20	52.66	14.0	41.0
med-replay	hopper	27.69	29.08	5.0	29.29	**31.29**	6.8	12.4
	halfcheetah	42.29	42.84	1.3	43.96	**44.25**	0.7	42.3
	walker2d	19.84	27.30	37.6	35.59	**41.16**	15.7	9.7

Table 1: Normalized scores for LOOP on the D4RL datasets comparing to the underlying offline RL algorithms and a baseline MBOP. LOOP improves the base algorithm across various types of datasets and environments.

For Offline RL, we benchmark the performance using the D4RL datasets [53]. We combine LOOP with two value-based offline RL algorithms: Critic Regularized Regression (CRR) [54] and Policy in Latent Action Space (PLAS) [32]. We use the original offline RL algorithms to train a value function from the static data and then use it as the terminal value function for LOOP. We use $\beta = 1$ in the trajectory prior of ARC (Section 5.1) in the offline RL setting to keep the policy conservative.

Baselines: In addition to the underlying offline RL algorithms, we also include recent work MBOP [33] as a baseline. MBOP uses a terminal value function which is an evaluation of the dataset policy. In contrast, LOOP uses a terminal value function trained with offline RL algorithms which is more optimal.

Performance: Table 1 presents the comparison of LOOP and the underlying offline RL algorithms. LOOP offers an average improvement of 15.91% over CRR and 29.49% over PLAS on the complete D4RL MuJoCo Locomotion dataset. Full results can be found in Appendix D.3. The results further highlight the benefit of the LOOP framework compared to the underlying model-free algorithms.

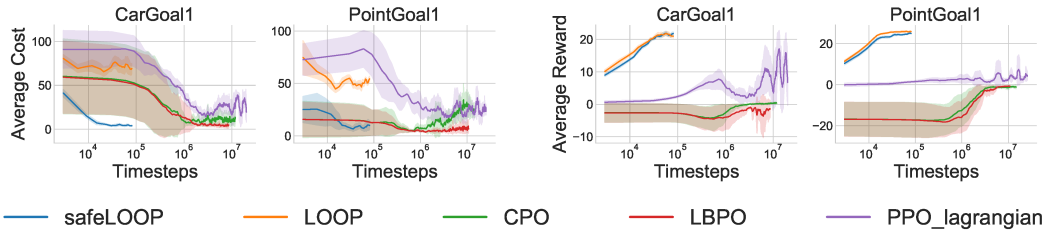

Figure 5: We compare safeLOOP with other safety methods such as CPO, LBPO, and PPO-lagrangian on OpenAI Safety Gym environments. It shows significant sample efficiency while offering similar or better safety benefits as the baselines.

6.3 LOOP for Safe RL

For safe RL, we modify the H-step lookahead optimization to maximize the sum of rewards while satisfying the cost constraints, as described in Section 5.2. We evaluate our method on two environments from the OpenAI Safety Gym [55] and an RC-car simulation environment [56]. The objective of the Safety Gym environments is to move a Point mass agent or a Car agent to the goal while avoiding obstacles. The RC-car environment is rewarded for driving along a circle of 1m fixed radius with a desired velocity while staying within the 1.2m circle during training. Details for the environments can be found in Appendix C.4.

Baselines: We compare our safety-augmented LOOP (safeLOOP) against various state-of-the-art safe learning methods such as CPO [57], LBPO [58], and PPO-lagrangian [59, 55]. CPO uses a trust region update rule that guarantees safety. LBPO relies on a barrier function formulated around a Lyapunov constraint for safety. PPO-lagrangian uses dual gradient descent to solve the constrained optimization. To ensure a fair comparison, all policies and dynamics models are randomly initialized, as is commonly done in safe RL experiments (rather than starting from a safe initial policy). We additionally compare against a model-based safety method that modifies PETS for safe exploration (safePETS) without the terminal value function. We mostly compare to model-free baselines due to a lack of safe model-based Deep-RL baselines in the literature.

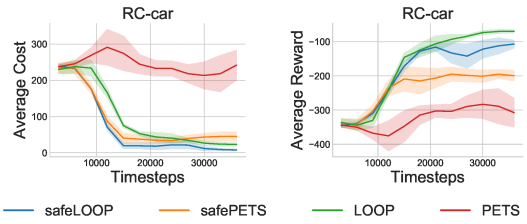

Figure 6: RC-car experiments show the importance of the terminal value function in the LOOP framework. SafeLOOP achieves higher returns than safePETS while being competitive in safety performance. Both safePETS and PETS fail to learn a drifting policy due to limited lookahead.

Performance: For the OpenAI Safety Gym environments, we observe in Figure 5 that safeLOOP can achieve performant yet safe policies in a sample efficient manner. SafeLOOP reaches a higher reward than CPO, LBPO and PPO-lagrangian, while being orders of magnitude faster. SafeLOOP also achieves a policy with a lower cost faster than the baselines. From another aspect, the simulated RC-car experiments demonstrate the benefits of the terminal value function in safe RL. Figure 6 shows the performance of LOOP, safeLOOP, PETS, and safePETS on this domain. PETS [17] and safePETS do not consider a terminal value function. SafeLOOP is able to achieve high performance while maintaining the fewest constraint violations during training. Qualitatively, LOOP and safeLOOP are able to learn a safe drifting behavior, whereas PETS and safePETS fail to do so since drifting requires longer horizon reasoning beyond the fixed planning horizon in PETS. The results suggest that safeLOOP is a desirable choice of algorithm for safe RL due to its sample efficiency and the flexibility of incorporating constraints during deployment.

7 Conclusion

In this work we analyze the H-step lookahead method under a learned model and value function and demonstrate empirically that it can lead to many benefits in deep reinforcement learning. We propose a framework LOOP which removes the computational overhead of trajectory optimization for value function update. We identify the actor-divergence issue in this framework and propose a modified trajectory optimization procedure - Actor Regularized Control. We show that the flexibility of H-step lookahead policy allows us to improve performance in online RL, offline RL as well as safe RL and this makes LOOP a strong choice of RL algorithm for robotic applications.

Acknowledgments

We thank Tejus Gupta, Xingyu Lin and the members of R-PAD lab for insightful discussions. This material is based upon work supported by the United States Air Force and DARPA under Contract No. FA8750-18-C-0092, LG Electronics, and the National Science Foundation under Grant No. IIS-1849154.

References

[1] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*, 2021.

[2] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[3] J. Matas, S. James, and A. J. Davison. Sim-to-real reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, pages 734–743. PMLR, 2018.

[4] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[5] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[6] S. Fujimoto, H. Van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[7] S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the Fourth Connectionist Models Summer School*, pages 255–263. Hillsdale, NJ, 1993.

[8] S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018.

[9] T. Lu, D. Schuurmans, and C. Boutilier. Non-delusional q-learning and value-iteration. 2018.

[10] A. Kumar, R. Agarwal, D. Ghosh, and S. Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. *arXiv preprint arXiv:2010.14498*, 2020.

[11] J. Fu, A. Kumar, M. Soh, and S. Levine. Diagnosing bottlenecks in deep q-learning algorithms. In *International Conference on Machine Learning*, pages 2021–2030. PMLR, 2019.

[12] J. Achiam, E. Knight, and P. Abbeel. Towards characterizing divergence in deep q-learning. *arXiv preprint arXiv:1903.08894*, 2019.

[13] M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization, 2019.

[14] A. Rajeswaran, I. Mordatch, and V. Kumar. A game theoretic framework for model based reinforcement learning. In *International Conference on Machine Learning*, pages 7953–7963. PMLR, 2020.

[15] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

[16] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, pages 8224–8234, 2018.

[17] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.

[18] Y. Efroni, M. Ghavamzadeh, and S. Mannor. Online planning with lookahead policies. *Advances in Neural Information Processing Systems*, 33, 2020.

[19] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. *arXiv preprint arXiv:1811.01848*, 2018.

[20] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

[21] A. Nagabandi, K. Konoglie, S. Levine, and V. Kumar. Deep dynamics models for learning dexterous manipulation. *arXiv preprint arXiv:1909.11652*, 2019.

[22] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

[23] S. Levine and V. Koltun. Guided policy search. In *International conference on machine learning*, pages 1–9. PMLR, 2013.

[24] I. Clavera, V. Fu, and P. Abbeel. Model-augmented actor-critic: Backpropagating through paths. *arXiv preprint arXiv:2005.08068*, 2020.

[25] R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

[26] A. Piché, V. Thomas, C. Ibrahim, Y. Bengio, and C. Pal. Probabilistic planning with sequential monte carlo methods. In *International Conference on Learning Representations*, 2018.

[27] R. Agarwal, D. Schuurmans, and M. Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

[28] R. Zhang, B. Dai, L. Li, and D. Schuurmans. Gendice: Generalized offline estimation of stationary values. *arXiv preprint arXiv:2002.09072*, 2020.

[29] N. Y. Siegel, J. T. Springenberg, F. Berkenkamp, A. Abdolmaleki, M. Neunert, T. Lampe, R. Hafner, and M. Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *arXiv preprint arXiv:2002.08396*, 2020.

[30] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[31] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[32] W. Zhou, S. Bajracharya, and D. Held. Plas: Latent action space for offline reinforcement learning. *arXiv preprint arXiv:2011.07213*, 2020.

[33] A. Argenson and G. Dulac-Arnold. Model-based offline planning. *arXiv preprint arXiv:2008.05556*, 2020.

[34] E. Altman. *Constrained Markov decision processes*, volume 7. CRC Press, 1999.

[35] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440. IEEE, 2016.

[36] R. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1(2):127–190, 1999.

[37] E. F. Camacho and C. B. Alba. *Model predictive control*. Springer science & business media, 2013.

[38] T. Wang and J. Ba. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*, 2019.

[39] B. Zhang, R. Rajan, L. Pineda, N. Lambert, A. Biedenkapp, K. Chua, F. Hutter, and R. Calandra. On the importance of hyperparameter optimization for model-based reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 4015–4023. PMLR, 2021.

[40] S. P. Singh and R. C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233, 1994.

[41] A. Agarwal, N. Jiang, and S. M. Kakade. Reinforcement learning: Theory and algorithms. *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep*, 2019.

[42] T. Matsushima, H. Furuta, Y. Matsuo, O. Nachum, and S. Gu. Deployment-efficient reinforcement learning via model-based offline optimization. *arXiv preprint arXiv:2006.03647*, 2020.

[43] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11761–11771, 2019.

[44] N. Vieillard, T. Kozuno, B. Scherrer, O. Pietquin, R. Munos, and M. Geist. Leverage the average: an analysis of regularization in rl. *arXiv preprint arXiv:2003.14089*, 2020.

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

[46] J. Peters and S. Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th international conference on Machine learning*, pages 745–750, 2007.

[47] J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[48] J. Peters, K. Mulling, and Y. Altun. Relative entropy policy search. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[49] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

[50] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims. Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*, 2020.

[51] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[52] Y. Luo, H. Xu, Y. Li, Y. Tian, T. Darrell, and T. Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. *arXiv preprint arXiv:1807.03858*, 2018.

[53] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[54] Z. Wang, A. Novikov, K. Żołna, J. T. Springenberg, S. Reed, B. Shahriari, N. Siegel, J. Merel, C. Gulcehre, N. Heess, et al. Critic regularized regression. *arXiv preprint arXiv:2006.15134*, 2020.

[55] A. Ray, J. Achiam, and D. Amodei. Benchmarking Safe Exploration in Deep Reinforcement Learning. 2019.

[56] E. Ahn. *Towards safe reinforcement learning in the real world*. PhD thesis, 2019.

[57] J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31. PMLR, 2017.

[58] H. Sikchi, W. Zhou, and D. Held. Lyapunov barrier policy optimization. *arXiv preprint arXiv:2103.09230*, 2021.

[59] E. Altman. Constrained markov decision processes with total cost criteria: Occupation measures and primal lp. *Mathematical methods of operations research*, 43(1):45–72, 1996.

[60] R. Munos. Error bounds for approximate value iteration. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 1006. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[61] R. Munos and C. Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(5), 2008.

[62] Z. Liu, H. Zhou, B. Chen, S. Zhong, M. Hebert, and D. Zhao. Safe model-based reinforcement learning with robust cross-entropy method. *arXiv preprint arXiv:2010.07968*, 2020.

[63] M. Wen and U. Topcu. Constrained cross-entropy method for safe reinforcement learning. *IEEE Transactions on Automatic Control*, 2020.

[64] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv preprint arXiv:1612.01474*, 2016.

A Theory

A.1 H-step lookahead with approximation error

We aim to show that H-step model-based lookahead policies are more robust to certain types of approximation errors than 1-step greedy policies given an approximate value function. We restate Theorem 1 here for convenience and then provide a proof.

Theorem 1. *(H-step lookahead policy) Suppose $\hat{M}$ is an approximate dynamics model such that $\max_{s,a} D_{TV}\left(M(.|s,a), \hat{M}(.|s,a)\right) \leq \epsilon_m$. Let $\hat{V}$ be an approximate value function such that $\max_s |V^*(s) - \hat{V}(s)| \leq \epsilon_v$. Let the reward function by bounded in [0,R_{max}] and $\hat{V}$ be bounded in [0,V_{max}]. Let ϵ_p be the suboptimality incurred in H-step lookahead optimization (Eqn. 3) such that $J^* - \hat{J} \leq \epsilon_p$, where J^* is the optimal return for the H-step optimization and $\hat{J}$ is the result of the suboptimal H-step optimization. Then the performance of the H-step lookahead policy $\pi_{H,\hat{V}}$ can be bounded as:*

$$J^{\pi^*} - J^{\pi_{H,\hat{V}}} \leq \frac{2}{1-\gamma^H}[C(\epsilon_m, H, \gamma) + \frac{\epsilon_p}{2} + \gamma^H \epsilon_v]$$

where

$$C(\epsilon_m, H, \gamma) = R_{\max} \sum_{t=0}^{H-1} \gamma^t t \epsilon_m + \gamma^H H \epsilon_m V_{max}$$

Proof. Assume we have an ϵ_v-approximate value function i.e $\|\hat{V} - V^*\|_\infty < \epsilon_v$ and we have an approximate transition model which satisfies $D_{TV}\left(M(.|s,a), \hat{M}(.|s,a)\right) \leq \epsilon_m$, similar to assumptions in [19, 18]. We analyze the optimality gap of the policy which uses an H-step lookahead optimization (Eqn. 3) with this approximate model and value function. First, we define some useful notations: let $\mathcal{M}$ be the MDP defined by $(\mathcal{S}, \mathcal{A}, M, r, s_0)$ which uses the ground truth dynamics M, state space $\mathcal{S}$, action space $\mathcal{A}$, reward function r and starting state s_0, and let $\hat{\mathcal{M}}$ be the MDP defined by $(\mathcal{S}, \mathcal{A}, \hat{M}, r, s_0)$ which uses the approximate dynamics model $\hat{M}$. Correspondingly, let $\mathcal{H}$ be an H-step finite horizon MDP given by $(\mathcal{S}, \mathcal{A}, M, r_{\text{mix}}, s_0)$ and let $\hat{\mathcal{H}}$ be an H-step finite horizon MDP given by $(\mathcal{S}, \mathcal{A}, \hat{M}, r_{\text{mix}}, s_0)$ where

$$r_{\text{mix}}(s_t, a_t) = \begin{cases} r(s,a) & \text{if } t < H \\ \hat{V}(s_H) & \text{if } t = H \end{cases} \tag{11}$$

We redefine $\pi_{H,\hat{V}}$ to be the policy obtained by repeatedly optimizing for the H-step lookahead objective (Eqn. 3) in $\hat{\mathcal{H}}$ and acting for H steps in $\mathcal{M}$. We do not consider the MPC setting for simplicity in proof i.e. the policy does not perform any replanning after taking its initial actions. We will use $\pi^*_{\mathcal{K}}$ denote the optimal policy for some MDP $\mathcal{K}$. Let $\hat{\tau}$ denote an H-step trajectory sampled by running $\pi^*_{\hat{\mathcal{H}}}$ in $\mathcal{M}$ and similarly τ is used to denote an H-step trajectory sampled by running $\pi^*_{\mathcal{H}}$ in $\mathcal{M}$. Let τ^* denote the H-step trajectory sampled by running $\pi^*_{\mathcal{M}}$ in $\mathcal{M}$. Let $p_{\hat{\tau}}$, p_τ and p_{τ^*} be the corresponding trajectory distributions. The performance gap we want to upper bound is given by:

$$J^{\pi^*} - J^{\pi_{H,\hat{V}}} = V^*(s_0) - V^{\pi_{H,\hat{V}}}(s_0) \tag{12}$$

$$= \mathbb{E}_{\tau^* \sim p_{\tau^*}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H V^*(s_H)\right] - \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H V^{\pi_{H,\hat{V}}}(s_H)\right] \tag{13}$$

$$= \mathbb{E}_{\tau^* \sim p_{\tau^*}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H V^*(s_H)\right] - \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H V^*(s_H)\right] \tag{14}$$

$$+ \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H V^*(s_H)\right] - \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H V^{\pi_{H,\hat{V}}}(s_H)\right] \tag{15}$$

$$= \mathbb{E}_{\tau^* \sim p_{\tau^*}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H V^*(s_H)\right] - \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H V^*(s_H)\right] \tag{16}$$

$$+ \gamma^H \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}[V^*(s_H) - V^{\pi_{H,\hat{V}}}(s_H)] \tag{17}$$

Since we have $|V^*(s) - \hat{V}(s)| \leq \epsilon_v \; \forall s$, we can bound the following expressions:

$$\mathbb{E}_{\tau^* \sim p_{\tau^*}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H V^*(s_H)\right] \leq \mathbb{E}_{\tau^* \sim p_{\tau^*}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] + \gamma^H \epsilon_v \quad (18)$$

$$\mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H V^*(s_H)\right] \geq \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] - \gamma^H \epsilon_v \quad (19)$$

Subtracting these two inequalities (18 and 19), we get:

$$\mathbb{E}_{\tau^* \sim p_{\tau^*}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H V^*(s_H)\right] - \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H V^*(s_H)\right] \quad (20)$$

$$\leq \mathbb{E}_{\tau^* \sim p_{\tau^*}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] - \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] + 2\gamma^H \epsilon_v$$

Substituting Eqn. 20 into Eqn. 16 we can bound the performance gap as follows:

$$J^{\pi^*} - J^{\pi_{H,\hat{V}}} = V^*(s_0) - V^{\pi_{H,\hat{V}}}(s_0)$$

$$\leq \mathbb{E}_{\tau^* \sim p_{\tau^*}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] - \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] \quad (21)$$

$$+ 2\gamma^H \epsilon_v + \gamma^H \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}[V^*(s_H) - V^{\pi_{H,\hat{V}}}(s_H)]$$

$$= \mathbb{E}_{\tau^* \sim p_{\tau^*}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] - \mathbb{E}_{\tau \sim p_{\tau}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] \quad (22)$$

$$+ \mathbb{E}_{\tau \sim p_{\tau}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] - \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right]$$

$$+ 2\gamma^H \epsilon_v + \gamma^H \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}[V^*(s_H) - V^{\pi_{H,\hat{V}}}(s_H)] \quad (23)$$

$$\leq \mathbb{E}_{\tau \sim p_{\tau}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] - \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] \quad (24)$$

$$+ 2\gamma^H \epsilon_v + \gamma^H \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}[V^*(s_H) - V^{\pi_{H,\hat{V}}}(s_H)] \quad (25)$$

The last step is due to the fact that τ is generated by the optimal action sequence in the *ground-truth* H-step MDP $\mathcal{H}$ as defined earlier which implies that $\mathbb{E}_{\tau^* \sim p_{\tau^*}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] \leq \mathbb{E}_{\tau \sim p_{\tau}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right]$

Now we aim to characterize the performance gap between an optimal policy of MDP $\hat{\mathcal{H}}$, $\pi^*_{\hat{\mathcal{H}}}$, with the optimal policy of MDP $\mathcal{H}$, $\pi^*_{\mathcal{H}}$, evaluating both in the ground truth MDP $\mathcal{H}$. We wish to characterize this performance gap as a function of model errors and value errors $f(\epsilon_m, \epsilon_v, \gamma, H)$:.

$$\mathbb{E}_{\tau \sim p_{\tau}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] - \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] \leq f(\epsilon_m, \epsilon_v, \gamma, H)$$

Let $J^{\pi}_{\mathcal{H}}$ denote the performance of policy π when evaluated in MDP $\mathcal{H}$ starting from same initial state s_0. Then we can write this performance gap as

$$\mathbb{E}_{\tau \sim p_{\tau}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] - \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H)\right] \quad (26)$$

$$= J^{\pi^*_{\mathcal{H}}}_{\mathcal{H}} - J^{\pi^*_{\hat{\mathcal{H}}}}_{\mathcal{H}} \quad (27)$$

$$= J^{\pi^*_{\mathcal{H}}}_{\mathcal{H}} - J^{\pi^*_{\mathcal{H}}}_{\hat{\mathcal{H}}} + J^{\pi^*_{\mathcal{H}}}_{\hat{\mathcal{H}}} - J^{\pi^*_{\hat{\mathcal{H}}}}_{\hat{\mathcal{H}}} + J^{\pi^*_{\hat{\mathcal{H}}}}_{\hat{\mathcal{H}}} - J^{\pi^*_{\hat{\mathcal{H}}}}_{\mathcal{H}} \quad (28)$$

$$= \left(J^{\pi^*_{\mathcal{H}}}_{\mathcal{H}} - J^{\pi^*_{\mathcal{H}}}_{\hat{\mathcal{H}}}\right) - \left(J^{\pi^*_{\hat{\mathcal{H}}}}_{\mathcal{H}} - J^{\pi^*_{\hat{\mathcal{H}}}}_{\hat{\mathcal{H}}}\right) + \left(J^{\pi^*_{\mathcal{H}}}_{\hat{\mathcal{H}}} - J^{\pi^*_{\hat{\mathcal{H}}}}_{\hat{\mathcal{H}}}\right) \quad (29)$$

$$\leq \left(J^{\pi^*_{\mathcal{H}}}_{\mathcal{H}} - J^{\pi^*_{\mathcal{H}}}_{\hat{\mathcal{H}}}\right) - \left(J^{\pi^*_{\hat{\mathcal{H}}}}_{\mathcal{H}} - J^{\pi^*_{\hat{\mathcal{H}}}}_{\hat{\mathcal{H}}}\right) + \epsilon_p \quad (30)$$

$$\leq 2 \max_{\pi \in \{\pi^*_{\mathcal{H}}, \pi^*_{\hat{\mathcal{H}}}\}} |\left(J^{\pi}_{\mathcal{H}} - J^{\pi}_{\hat{\mathcal{H}}}\right)| + \epsilon_p \quad (31)$$

The second-to-last equation is due to the assumed suboptimality of H-step lookahead planner where we have $\forall$ policies π, $J^{\pi^*_{\hat{\mathcal{H}}}}_{\hat{\mathcal{H}}} + \epsilon_p \geq J^{\pi}_{\hat{\mathcal{H}}}$. Since the total variation between M and $\hat{M}$ is at most ϵ_m, i.e $D_{TV}\left(M(.|s,a), \hat{M}(.|s,a)\right) \leq \epsilon_m$, we have that $|\rho^t_1(s,a) - \rho^t_2(s,a)| \leq t\epsilon_m$, where $\rho_1(s,a)$

is the discounted state-action visitation induced by π on $\mathcal{H}$, $\rho_2(s,a)$ is the discounted state-action visitation induced by the same policy on $\hat{\mathcal{H}}$ and superscript t indicates the state-action marginal at the t^{th} timestep (for proof see Lemma B.2 Markov Chain TVD Bound [13]). Then we can write the performance of policy π in terms of its induced state marginal and the reward function, i.e $J^\pi_{\mathcal{H}} = \sum_{s,a} \rho_1(s,a) r_{\text{mix}}(s,a) = \sum_{s,a} \sum_{t=0}^{H} \gamma^t \rho_1^t(s,a) r_{\text{mix}}(s,a)$ and use the Markov chain TVD bound:

$$J^\pi_{\mathcal{H}} - J^\pi_{\hat{\mathcal{H}}} = \sum_{s,a} (\rho_1(s,a) - \rho_2(s,a)) r_{\text{mix}}(s,a) \tag{32}$$

$$|J^\pi_{\mathcal{H}} - J^\pi_{\hat{\mathcal{H}}}| = |\sum_{s,a} (\rho_1(s,a) - \rho_2(s,a)) r_{\text{mix}}(s,a)| \tag{33}$$

$$= |\sum_{s,a} \sum_{t=0}^{H} \gamma^t (\rho_1^t(s,a) - \rho_2^t(s,a)) r^t_{\text{mix}}(s,a)| \tag{34}$$

$$\leq \sum_{s,a} \sum_{t=0}^{H} \gamma^t |(\rho_1^t(s,a) - \rho_2^t(s,a))| r^t_{\text{mix}}(s,a) \tag{35}$$

$$\leq R_{\max} \sum_{t=0}^{H-1} \gamma^t t \epsilon_m + \gamma^H H \epsilon_m V_{\max} \tag{36}$$

$$= C(\epsilon_m, H, \gamma) \tag{37}$$

Combining Eqn. 31 and Eqn. 37 we have:

$$\mathbb{E}_{\tau \sim p_\tau} \left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H) \right] - \mathbb{E}_{\hat{\tau} \sim p_{\hat{\tau}}} \left[\sum \gamma^t r(s_t, a_t) + \gamma^H \hat{V}(s_H) \right] \leq 2C(\epsilon_m, H, \gamma) + \epsilon_p \tag{38}$$

We substitute Eqn. 38 in Eqn. 24. Also observe that the last term in Eqn. 24 $\gamma^H \mathbb{E}_{\hat{\tau}}[V^*(s_H) - V^{\pi_{H,\hat{V}}}(s_H)]$ can be bounded recursively. Then, we will have the following optimality gap for the H-step lookahead policy $\pi_{H,\hat{V}}$:

$$J^{\pi^*} - J^{\pi_{H,\hat{V}}} \leq \frac{2}{1-\gamma^H} [C(\epsilon_m, H, \gamma) + \frac{\epsilon_p}{2} + \gamma^H \epsilon_v] \tag{39}$$

The H-step lookahead policy $\pi_{H,\hat{V}}$ reduces the dependency on ϵ_v (the maximum error of the value function) by a factor of γ^H and introduces an additional dependency on ϵ_m (the maximum error of the model). In contrast, when we use 1-step greedy policy, the performance gap is bounded by (Lemma 1):

$$J^{\pi^*} - J^{\pi_{H,\hat{V}}} \leq \frac{\gamma}{1-\gamma} [2\epsilon_v] \tag{40}$$

Lemma 1 can be seen as a special case of our bound when ϵ_m is set to 0 and H is set to 1. □

A.2 H-step lookahead with model generalization error

In this section, we derive a similar proof as the previous section with a weaker assumption on model error. We consider a model trained by supervised learning where the sample error can be computed by PAC generalization bounds which bounds the expected loss and empirical loss under a dataset with high probability.

We define D to be the dataset of transitions and π_D to be the data collecting policy.

Corollary 1. *(H-step lookahead with function approximation) Suppose $\hat{M}$ is an approximate dynamics model such that $\max_t \mathbb{E}_{s \sim \pi_{D,t}} \left[D_{TV}(M(.|s,a) \| \hat{M}(.|s,a)) \right] \leq \tilde{\epsilon}_m$. Let $\hat{V}$ be an approximate value function such that $\max_s |V^*(s) - \hat{V}(s)| \leq \epsilon_v$. Let the maximum TV distance of state distribution visited by lookahead policy $\pi_{H,\hat{V}}$ be bounded wrt state visitation of data generating policy by $\max_t \mathbb{E}_{s \sim \pi_{H,\hat{V}}} \left[D_{TV}(\rho^t_{\pi_{H,\hat{V}}} \| \rho^t_{\pi_D}) \right] \leq \epsilon_i$ and $\max \left(D_{TV}(\pi_D(a|s) || \pi^*_H(a|s)), D_{TV}(\pi_D(a|s) || \pi^*_{\hat{H}}(a|s)) \right) \leq \tilde{\epsilon_\pi} \ \ \forall s$. Let the reward function by*

bounded in [0,R_{max}] and $\hat{V}$ be bounded in [0,V_{max}]. Then the performance of the H-step lookahead policy $\pi_{H,\hat{V}}$ can be bounded as:

$$J^{\pi^*} - J^{\pi_{H,\hat{V}}} \leq \frac{2}{1-\gamma^H}[C(\tilde{\epsilon_m}, \tilde{\epsilon_\pi}, \epsilon_i, H, \gamma) + \gamma^H \epsilon_v]$$

where

$$C(\tilde{\epsilon_m}, \tilde{\epsilon_\pi}, H, \gamma) = R_{\max} \sum_{t=0}^{H-1} \gamma^t t(\tilde{\epsilon_m} + \tilde{\epsilon_\pi}) + R_{\max}\epsilon_i + \gamma^H H(\tilde{\epsilon_m} + \tilde{\epsilon_\pi})V_{max}$$

Proof. In the function approximation setting, a more realistic perfomance bound depends on the generalization error of model and distribution shift for the new policy under the collected dataset of transitions D. Let π_D be the data collecting policy. Let $\mathbb{E}_{s\sim\pi_{D,t}}\Big[D_{TV}(M(.|s,a)||\hat{M}(.|s,a))\Big] \leq \tilde{\epsilon}_m \ \forall s$ and $\max\Big(D_{TV}(\pi_D(a|s)||\pi^*_H(a|s)), D_{TV}(\pi_D(a|s)||\pi^*_{\hat{H}}(a|s))\Big) \leq \tilde{\epsilon_\pi} \ \forall s$. Following Lemma B.2 Markov Chain TVD Bound [13] with model generalization error $\tilde{\epsilon}_m$, policy distribution shift $\tilde{\epsilon}_\pi$ and bounded state visitation of lookahead policy by ϵ_i, we have: $|\rho^t_1(s,a) - \rho^t_2(s,a)| \leq t(\tilde{\epsilon}_m + \tilde{\epsilon}_\pi) + \epsilon_i$ Substituting the new state-action divergence bound in Eqn. 35 from Theorem 1 we get the following performance bound:

$$J^{\pi^*} - J^{\pi_{H,\hat{V}}} \leq \frac{2}{1-\gamma^H}[C(\tilde{\epsilon_m}, \tilde{\epsilon_\pi}, \epsilon_i, H, \gamma) + \gamma^H \epsilon_v] \tag{41}$$

where $C(\tilde{\epsilon_m}, \tilde{\epsilon_\pi}, H, \gamma) = R_{\max} \sum_{t=0}^{H-1} \gamma^t t(\tilde{\epsilon_m} + \tilde{\epsilon_\pi}) + R_{\max}\epsilon_i + \gamma^H H(\tilde{\epsilon_m} + \tilde{\epsilon_\pi})V_{\max}$.

Intuitively this bound highlights the tradeoff between model error and value error reasonably when the dataset is sufficiently exploratory to cover π^*_H and H-step lookahead policy has visitation close to the dataset.

□

A.3 H-step lookahead with Empirical Dataset Distribution using Fitted-Q Iteration

In this section, we take a look at the analysis of H-step lookahead under a set of different assumptions. In particular, we assume a form of model generalization error and that the optimal H-step trajectory is obtained via fitted-Q iteration in the H-step MDP at every timestep during policy deployment. This analysis largely follows the fitted-Q iteration analysis from [60, 61, 41] but we adapt it to H-step lookahead in a simplified form.

Assumption 1. *Let our replay buffer dataset be denoted by D and the data generating distribution be given by d^{π_D}, where π_D is the data generating policy. Let the Q-function class is given by $\mathcal{Q} \subset \mathbb{R}^{S\times A}$. The empirical bellman update $\hat{\mathcal{T}}Q$ under the learned model is given by:*

$$L_{d^{\pi_{\hat{M}}}}(Q, Q^k) = \mathbb{E}_{s,a,r,s'\sim d^{\pi_{\hat{M}}}}\left[(Q(s,a) - r - \gamma Q^k(s', \pi_Q(s')))^2\right] \tag{42}$$

where Q^k is the Q-function at k iteration, $d^{\pi_{\hat{M}}}$ is the state visitation under a learned model $\hat{M}$ from dataset D. Also we define:

$$L_{d^{\pi_D}}(Q, Q^k) = \mathbb{E}_{s,a,r,s'\sim d^{\pi_D}}\left[(Q(s,a) - r - \gamma Q^k(s', \pi_Q(s')))^2\right] \tag{43}$$

A form of model generalization error: We assume the following uniform deviation bound which holds with high probability ($\geq 1-\delta$):

$$\forall Q, Q^k, \ |L_D(Q, Q^k) - L_{d^{\pi_D}}(Q, Q^k)| \leq \tilde{\epsilon}_m \tag{44}$$

This bound can be obtained by concentration inequality as in [41] using concentration inequality $\tilde{\epsilon}_m$ to be a function of size of dataset $|D|$, δ and size of function space for $\mathcal{Q}$.

Intuitively the assumption above states that the bellman error obtained in the data-generating distribution is close to the bellman error obtained via state-action distribution induced by the learned model, where the model is learned on a finite fixed dataset D sampled from data generating distribution.

In the following analysis, we assume that H-step lookahead policy is obtained by performing fitted-Q iteration in the H-step approximate MDP $\hat{\mathcal{H}}$ defined in Theorem 1.

Theorem 2. *Suppose $\hat{M}$ is an approximate dynamics model such that Assumption 1 holds. Let $\hat{V}$ be an approximate value function such that $\max_s |V^*(s) - \hat{V}(s)| \leq \epsilon_v$. Let the reward function by bounded in [0,R_{max}] and $\hat{V}$ be bounded in [0,V_{max}]. Let concentrability coefficient $\tilde{C}$ be such that $\forall s, a \ \frac{\nu(s,a)}{d^{\pi_D}(s,a)} \leq \tilde{C}$ where $\nu(s,a)$ is state-action distribution induced by any non-stationary policy. Then the performance of the H-step lookahead policy $\pi_{H,\hat{V}}$ obtained by running fitted-Q iteration on the learned model to convergence can be bounded as:*

$$J^{\pi^*} - J^{\pi_{H,\hat{V}}} \leq \frac{2}{1-\gamma^H}[C(\tilde{\epsilon}_m, \tilde{C}, H, \gamma) + \gamma^H \epsilon_v]$$

where

$$C(\tilde{\epsilon}_m, \tilde{C}, H, \gamma) = \frac{2(1-\gamma^H)}{1-\gamma}\left(\frac{1}{1-\gamma}\sqrt{2\tilde{\epsilon}_m \tilde{C}}\right)$$

Proof. In this section we analyze the performance of H-step lookahead policies under the assumptions for Fitted Q Iteration [41]. This analysis extends the fitted-Q iteration analysis from greedy to H-step lookahead policies.

Let $\|g\|_{p,\nu}$ denote a weighted p-norm under distribution ν given by $\|g\|_{p,\nu} = \mathbb{E}_{s\sim\nu}[|g(s)|^p]^{\frac{1}{p}}$. We start by reusing the previous analysis in Theorem 1 under the new stated assumptions to replace the bound for Eqn. 26. Let $\pi^*_{\hat{\mathcal{H}}}$ be denoted by $\hat{\pi}_H$ and $\pi^*_{\mathcal{H}}$ by π^*_H for ease of notation. In this analysis $\hat{\pi}_H$ is the 1-step greedy policy obtained from Q_k the learned Q-function after k iterations of fitted-Q iteration on the H-step MDP $\hat{\mathcal{H}}$.

Rewriting Eqn. 26:

$$\mathbb{E}_{\tau\sim p_\tau}\left[\sum \gamma^t r(s_t,a_t) + \gamma^H \hat{V}(s_H)\right] - \mathbb{E}_{\hat{\tau}\sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t,a_t) + \gamma^H \hat{V}(s_H)\right] \tag{45}$$

$$= J^{\pi^*_H}_{\mathcal{H}} - J^{\hat{\pi}_H}_{\mathcal{H}} \tag{46}$$

Using performance difference lemma we can write:

$$J^{\pi^*_H}_{\mathcal{H}} - J^{\hat{\pi}_H}_{\mathcal{H}} \leq \sum_{t=1}^{H} \gamma^{t-1} \mathbb{E}_{s\sim d^{\hat{\pi}_H}}\left[V^{\pi^*_H}(s) - Q^{\pi^*_H}(s, \hat{\pi}_H)\right] \tag{47}$$

$$\leq \sum_{t=1}^{H} \gamma^{t-1} \mathbb{E}_{s\sim d^{\hat{\pi}_H}}\left[V^{\pi^*_H}(s) - Q_k(s,\pi^*_H) + Q_k(s,\hat{\pi}_H) - Q^{\pi^*_H}(s,\hat{\pi}_H)\right] \tag{48}$$

$$\leq \sum_{t=1}^{H} \gamma^{t-1} \left(\|Q^{\pi^*_H} - Q_k\|_{1,d^{\hat{\pi}_H}\times\pi^*_H} + \|Q^{\pi^*_H} - Q_k\|_{1,d^{\hat{\pi}_H}\times\hat{\pi}_H}\right) \tag{49}$$

$$\leq \sum_{t=1}^{H} \gamma^{t-1} \left(\|Q^{\pi^*_H} - Q_k\|_{d^{\hat{\pi}_H}\times\pi^*_H} + \|Q^{\pi^*_H} - Q_k\|_{d^{\hat{\pi}_H}\times\hat{\pi}_H}\right) \tag{50}$$

The second line follows from the fact that $Q_k(s,\hat{\pi}_H) \geq Q_k(s,\pi^*_H)$ since $\hat{\pi}_H$ maximizes Q_k.The concentrability assumptions allows us to compare weighted norms under state distribution induced by any policy $\nu(s,a)$ and $d^{\pi_D}(s,a)$ as follows: $\|.\|_\nu \leq \sqrt{\tilde{C}}\|.\|_{d^{\pi_D}}$. We can bound $\|Q^{\pi^*_H} - Q_k\|_{\mu,\pi}$ for arbitrary state distribution μ and policy π as:

$$\|Q^{\pi^*_H} - Q_k\|_{\mu\times\pi} = \|Q^{\pi^*_H} - \mathcal{T}Q_{k-1} + \mathcal{T}Q_{k-1} - Q_k\| \tag{51}$$

$$\leq \|\mathcal{T}Q^{\pi^*_H} - \mathcal{T}Q_{k-1}\|_{\mu\times\pi} + \|\mathcal{T}Q_{k-1} - Q_k\|_{\mu\times\pi} \tag{52}$$

$$\leq \|\mathcal{T}Q^{\pi^*_H} - \mathcal{T}Q_{k-1}\|_{\mu\times\pi} + \sqrt{\tilde{C}}\|\mathcal{T}Q_{k-1} - Q_k\|_{d^{\pi_D}} \tag{53}$$

$$= \gamma\|Q_{k-1}(\cdot,\pi_{Q_{k-1}}) - Q^{\pi^*_H}(\cdot,\pi^*_H)\|_{P(\mu\times\pi)} + \sqrt{\tilde{C}}\|\mathcal{T}Q_{k-1} - Q_k\|_{d^{\pi_D}} \tag{54}$$

where $P(\mu \times \pi)$ as distribution over $\mathcal{S}$ where $s, a \sim \mu$, $s' \sim p(s,a)$. Define $\pi_{mix} = \text{argmax}_{a \in \mathcal{A}}(Q^{\pi^*_H}(s,a), Q_{k-1}(s,a))$. Then we have:

$$\|Q^{\pi^*_H} - Q_k\|_{\nu,\pi} = \gamma \|Q_{k-1}(.,\pi_{Q_{k-1}}) - Q^{\pi^*_H}(.,\pi^*_H)\|_{P(\mu\times\pi)} + \sqrt{|\mathcal{A}|\tilde{C}}\|\mathcal{T}Q_{k-1} - Q_k\|_{d^{\pi_D}} \quad (55)$$

$$\leq \sqrt{\tilde{C}}\|\mathcal{T}Q_{k-1} - Q_k\|_{d^{\pi_D}} + \gamma \|Q_{k-1} - Q^{\pi^*_H}\|_{P(\mu\times\pi)\times\pi_{mix}} \quad (56)$$

The second term $\|Q_{k-1} - Q^{\pi^*_H}\|_{P(\mu\times\pi)\times\pi_{mix}}$ can be expanded via recursion for k times, since the same analysis holds. We now bound $\|\mathcal{T}Q_{k-1} - Q_k\|_{d^{\pi_D}}$.

$$\|\mathcal{T}Q_{k-1} - Q_k\|^2_{d^{\pi_D}} = L_{d^{\pi_D}}(Q_k, Q_{k-1}) - L_{d^{\pi_D}}(\mathcal{T}Q_{k-1}, Q_{k-1}) \quad (57)$$

$$\leq L_D(Q_k, Q_{k-1}) - L_D(\mathcal{T}Q_{k-1}, Q_{k-1}) + 2\tilde{\epsilon}_m \;\; \text{w.p} \geq 1-\delta \quad (58)$$

$$\leq 2\tilde{\epsilon}_m \quad (59)$$

As fitted Q iteration converges $k \to \infty$ for $\gamma < 1$, we have:

$$\|Q^{\pi^*_H} - Q_k\|_{\mu\times\pi} \leq \frac{1-\gamma^k}{1-\gamma}\sqrt{2\tilde{\epsilon}_m\tilde{C}} + \gamma^k \frac{V_{max}}{1-\gamma} \quad (60)$$

In this analysis we obtain $\hat{\pi}_H$ by performing fitted Q iteration ($k \to \infty$) under the dataset D. Therefore our bound for Eqn. 26 from the previous analysis under the current assumptions reduces to:

$$\mathbb{E}_{\tau\sim p_\tau}\left[\sum \gamma^t r(s_t,a_t) + \gamma^H \hat{V}(s_H)\right] - \mathbb{E}_{\hat{\tau}\sim p_{\hat{\tau}}}\left[\sum \gamma^t r(s_t,a_t) + \gamma^H \hat{V}(s_H)\right] \quad (61)$$

$$= J^{\pi^*_H}_{\mathcal{H}} - J^{\hat{\pi}_H}_{\mathcal{H}} \quad (62)$$

$$\leq \frac{2(1-\gamma^H)}{1-\gamma}\left(\frac{1}{1-\gamma}\sqrt{2\tilde{\epsilon}_m\tilde{C}}\right) \quad (63)$$

$$\leq C(\tilde{\epsilon}_m, \tilde{C}, H, \gamma) \quad (64)$$

Plugging this back in our previous analysis we have the following performance bound for H-step lookahead policy:

$$J^{\pi^*} - J^{\pi_{H,\hat{V}}} \leq \frac{2}{1-\gamma^H}[C(\tilde{\epsilon}_m, \tilde{C}, H, \gamma) + \gamma^H \epsilon_v] \quad (65)$$

where $C(\tilde{\epsilon}_m, \tilde{C}, H, \gamma) = \frac{2(1-\gamma^H)}{1-\gamma}\left(\frac{1}{1-\gamma}\sqrt{2\tilde{\epsilon}_m\tilde{C}}\right)$. □

A.4 ARC constrains trajectories close to the parameterized actor

In section 5.1, we use ARC, an iterative importance sampling procedure to solve the constrained optimization in Eqn. 6. The following lemma shows that the final trajectory distribution output as a result of finite importance sampling iteration is bounded in total variation to the trajectory distribution given by the parameterized actor.

Lemma 2. *Let p^τ_{prior} be a distribution over action sequences. Applying M KL-based trust region steps of size ϵ to p^τ_{prior} results in a distribution p^τ_M that satisfies:*

$$D_{TV}(p^\tau_{prior}||p^\tau) \leq T\sqrt{\frac{\epsilon}{2}} \quad (66)$$

Proof. This lemma is adapted from [42] and provided for completeness. Let p^τ_k be the distribution at the k trust region step. $p^\tau_0 = p^\tau_{prior}$ Using Pinsker's inequality we have:

$$D_{KL}(p^\tau_k||p^\tau_{k+1}) \leq \epsilon \quad (67)$$

$$D_{TV}(p^\tau_k||p^\tau_{k+1}) \leq \sqrt{\frac{\epsilon}{2}} \quad (68)$$

Using triangle inequality we have:

$$D_{TV}(p^\tau_{prior}||p^\tau_M) \leq M\sqrt{\frac{\epsilon}{2}} \quad (69)$$

□

B Algorithm Details

B.1 LOOP for online RL

Algorithm 1 LOOP-SAC (for Online RL and Safe RL)

Initialize the parametrized actor π_ϕ, Q-function Q_θ, predictive model $\hat{M}_\psi$, empty replay buffer D. Given planning horizon H.

// Training
for $t = 1..(train_steps)$ **do**
    Select action given by $a = ARC(s, \pi_\phi)$. ▷ Use safeARC for safeLOOP
    Execute a in the environment and observe reward r and new state s'.
    Store the transition (s, a, r, s') in replay buffer D.
    Optimize π_ϕ and Q_θ using SAC over replay buffer D.
    Train model $\hat{M}_\psi$ on the replay buffer D until convergence every K_m training steps.
end for
// Evaluation
for $t = 1..(eval_steps)$ **do**
    Select action given by $a = ARC(s, \pi_\phi)$.
    Execute a in the environment and observe reward r and new state s'.
end for

B.2 LOOP for offline RL

Algorithm 2 LOOP-offline

Initialize the parametrized actor π_ϕ, Q-function Q_θ, predictive model $\hat{M}_\psi$, empty replay buffer D. Given planning horizon H.

// Training
Train model $\hat{M}_\psi$ on the replay buffer D till convergence.
Run an Offline RL algorithm till convergence on D to learn Q_θ and π_ϕ.
// Evaluation
for $t = 1..(eval_steps)$ **do**
    Select action given by $a = ARC(s_t, \pi_\phi)$.
    Execute a in the environment and observe reward r and new state s'.
end for

B.3 Actor Regularized Control (ARC)

Eqn. 6 gives a general constrained optimization for policy update. In Eqn. 6, with terminal state-action value functions,when $[p^{\tau}_{prior} = \text{Uniform} , H = 0]$, we recover the SAC [5] deployment policy, when $[p^{\tau}_{prior} = \pi^{\beta} , H = 0]$, we recover the AWAC [45] deployment policy and when $\pi_{prior} = \mathcal{N}(0, \sigma)$, we recover the MPPI [35] deployment policy.

In the LOOP framework we use ARC as our trajectory optimization routine to solve Eqn. 6. Algorithm 3 shows the pseudocode for ARC routine used for Online and Offline RL experiments.

Algorithm 3 Actor Regularized Control (ARC)

Input: $s_T,\ \pi_\phi$

Given the parameterized actor π_ϕ, Q-function Q_θ, predictive model $\hat{M}_\psi$, reward model $\hat{r}$, replay buffer D, Planning Horizon H, 1-timestep shifted solution from the previous timestep μ^{T-1}, ARC iterations n_{ARC}, number of trajectories (population size) N.

for $i = 1..n_{ARC}$ **do**
  $\mathbf{R}_{1:N} = 0$ ▷ Rewards of N trajectories
  $\mathbf{A}_{1:N,1:H} = 0$ ▷ N action sequences with horizon H
  for $j = 1..N$ trajectories **do**
    // Generate a trajectory with the model
    $s_1 = s_T$
    for $t = 1..H$ horizon **do**
      // Generate actions from a mixture prior
      $\mathbf{A}_{j,t} = a_t = \beta\pi_\phi(s_t) + (1-\beta)\mathcal{N}(\mu^{T-1}_t, \sigma)$
      $s_{t+1} = \hat{M}_\psi(s_t, a_t)$
    end for
    // Rollout the action sequence P times in each model within the ensemble
    $R = 0$
    for $k = 1..K$models **do**
      for $p = 1..P$ particles **do**
        $s_1 = s_T$
        for $t = 1..H$ horizon **do**
          $a_t = \mathbf{A}_{j,t}$
          $s_{t+1} = \hat{M}_\psi(s_t, a_t)$
          $R = R + \gamma^{t-1}(\mathbb{1}(t = H)Q_\theta(s_t, a_t) + \mathbb{1}(t \neq H)\hat{r}(s_t, a_t))$
        end for
      end for
    end for
    //Uncertainty penalized average reward
    $\mathbf{R}_j = \frac{1}{K}\left(\sum_{k=1}^{K}(R/P) - \beta_{pess}\sum(R/P - \sum_{k=1}^{K}(\frac{R}{KP}))^2\right)$
  end for
  $\mu_{new,1:H}$ = weighted-mean($\mathbf{A}_{1:N}$, weights = exp($\mathbf{R}_{1:N}/\eta$))
  $\Sigma_{new,1:H}$ = weighted-mean($(\mathbf{A}_{1:N} - \mu_{new})^2$, weights = exp($\mathbf{R}_{1:N}/\eta$))
  $\mu^T_{i+1} = \ \alpha * \mu_{new} + (1-\alpha)\mu^T_i$ ▷ Update mean
  $\Sigma^T_{i+1} = \ \alpha * \Sigma_{new} + (1-\alpha)\Sigma^T_i$ ▷ Update variance
end for

Output: $\mu^T = \mu^T_{n_{ARC}+1}$

β_{pess} is set to zero for Online RL experiments and safe RL experiments where trajectories are scored by unpenalized average. It is tuned for Offline RL experiments as detailed in Appendix C.

B.3.1 ARC for safe-RL

We optimize for the following objective in LOOP for safe RL:

$$\text{argmax}_{a_t} \mathbb{E}_{\hat{M}}\left[R_{H,\hat{V}}(s_t)\right] \text{s.t. } \max_{[K]} \sum_{t=0}^{H} \gamma^t c(s_t, a_T) \leq d_0 \tag{70}$$

where $[K]$ are the model ensembles, c is the constraint cost function and $R_{H,\hat{V}}$ is the H-horizon lookahead objective defined in Eqn. 3. We incorporate safety in the trajectory optimization procedure following previous work [62, 63]. The pseudocode for safeARC used in safeLOOP is shown in Algorithm 4.

B.4 Discussion on the choice of terminal value function

LOOP-SAC, LOOP-SARSA and POLO use different ways to learn a terminal value funcion. LOOP-SARSA is evaluating the "replay buffer policy" instead of the H-step lookahead policy because we are using off-policy data (where the original SARSA is an on-policy algorithm). We believe this is the main reason behind its poor performance. Unfortunately, on-policy LOOP-SARSA would be too slow, due to the need for collecting on-policy data. POLO is formulated to evaluate V^π with the model. However, POLO requires running trajectory optimization during the value function update, which is computationally expensive. In contrast to these methods, LOOP uses an off-policy algorithm to learn V^*. We found that this approach has good performance and it is significantly more computationally efficient than POLO. An interesting direction of future work could be to try to combine LOOP with an efficient off-policy evaluation algorithm to estimate V^π.

Algorithm 4 safeARC

Input: $s_T,\ \pi_\phi$

Given the parameterized actor π_ϕ, Q-function Q_θ, predictive model $\hat{M}_\psi$, reward model $\hat{r}$, replay buffer D, planning horizon H, 1 timestep shifted solution from the previous timestep μ^{T-1}, safety threshold d_0, minimal safe trajectories m, ARC iterations n_{ARC}, number of trajectories (population size) N.

for $i = 1..n_{ARC}$ **do**
  $\mathbf{R}_{1:N} = 0$ ▷ Rewards of N trajectories
  $\mathbf{C}_{1:N} = 0$ ▷ Cost of N trajectories
  $\mathbf{A}_{1:N,1:H} = 0$ ▷ N action sequences with horizon H
  for $j = 1..N$ trajectories **do**
    // Generate a trajectory with the model
    $s_1 = s_T$
    for $t = 1..H$ horizon **do**
      // Generate actions from a mixture prior
      $\mathbf{A}_{j,t} = a_t = \beta\pi_\phi(s_t) + (1-\beta)\mathcal{N}(\mu_t^{T-1}, \sigma)$
      $s_{t+1} = \hat{M}_\psi(s_t, a_t)$
    end for
    // Rollout the action sequence P times in each model within the ensemble
    $R = 0$
    for $k = 1..K$ models **do**
      for $p = 1..P$ particles **do**
        $s_1 = s_T$
        for $t = 1..H$ horizon **do**
          $a_t = \mathbf{A}_{j,t}$
          $s_{t+1} = \hat{M}_\psi(s_t, a_t)$
          $R = R + \gamma^{t-1}(\mathbb{1}(t = H)Q_\theta(s_t, a_t) + \mathbb{1}(t \neq H)\hat{r}(s_t, a_t))$
          $C = C + \gamma^{t-1}(\hat{c}(s_t, a_t))$
        end for
      end for
    end for
    $\mathbf{R}_j = \frac{1}{K}\sum_{k=1}^{K}(R/P)$ ▷ Average Reward across the ensemble
    $\mathbf{C}_j = \max_{[K]}\max_{[P]}(C)$ ▷ Maximum Cost across the ensemble and particles
  end for
  if count$(\mathbf{C}_{1:N} < d_0) < m$ **then**
    μ_{new} = weighted-mean($\mathbf{A}_{1:N}$, weights = exp($-\mathbf{C}_N/\eta$))
    Σ_{new} = weighted-mean($(\mathbf{A}_{1:N} - \mu_{new})^2$, weights = exp($-\mathbf{C}_N/\eta$))
    ▷ Weighted mean w.r.t neg-cost
  else
    safe-idx $= \{i$ for $\mathbf{C}_i < d_0\}$
    μ_{new}= weighted-mean($\mathbf{A}_{\text{safe-idx}}$, weights = exp($\mathbf{R}_{\text{safe-idx}}/\eta$))
    Σ_{new} = weighted-mean($\mathbf{A}_{\text{safe-idx}} - \mu_{new})^2$, weights = exp($\mathbf{R}_{\text{safe-idx}}/\eta$))
    ▷ Weighted mean w.r.t safe actions
  end if
  $\mu_{i+1}^T = \ \alpha * \mu_{new} + (1-\alpha)\mu_i^T$ ▷ Update mean
  $\Sigma_{i+1}^T = \ \alpha * \Sigma_{new} + (1-\alpha)\Sigma_i^T$ ▷ Update variance
end for

Output: $\mu^T = \mu_{n_{ARC}+1}^T$

C Experiment Details

We use the same hyperparameters for the underlying off-policy method (SAC) and the ensemble dynamics models following previous work for LOOP and all the baselines [21, 17, 5]. All the results presented are averaged over 5 random seeds.

C.1 Implementation Details for the Dynamics Model Ensemble

Following [21, 17], we use probabilistic ensembles of dynamics models that capture the epistemic uncertainty as well as the aleatoric uncertainty in forward predictions [64]. The dynamics model $\hat{M}$ is comprised of K neural networks. Each individual network is randomly initialized and trained with the same dataset. Using the transition dataset, we train the dynamics model to predict the next state as well as the reward. In practice, instead of directly regressing to the next state, we instead predict Δ_{t+1}, where $\Delta_{t+1} = s_{t+1} - s_t$ parametrized as a Gaussian distribution with a diagonal covariance matrix. We regress directly to the scalar reward.

C.2 Online RL

Additional details on PenGoal-v1 and Claw-v1: We modify the original Pen-v1 environment [2] to have a narrower range of goals given by: $[0.7, 0.7] + \mathcal{N}(0, 0.1)$ and name this environment as PenGoal-v1. We use the Claw-v1 environment from Nagabandi et al. [21] using the original implementation[3] but we find the scale of rewards to be different from the paper.

Baselines: We use the original implementation for MBPO[4]. For SAC, we use a public implementation [5]. We use a planning horizon of 3 for PETS-restricted which is the same as LOOP. LOOP-SARSA is based on the same H-step lookahead idea, but with a terminal value function that is a evaluation of the replay buffer. The value function is updated using the following SARSA update from the replay buffer transitions:

$$\mathcal{T}^{\pi_D} Q(s_t, a_t) = r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}) \text{ , where } (s_t, a_t, r_t, s_{t+1}, a_{t+1} \sim D) \tag{71}$$

This baseline is similar to MBOP [33]. The main difference is that in this case the Q-function is learned via TD-backups for evaluation whereas MBOP uses Monte Carlo Evaluation. For SAC-VE, we implement H-step value expansion from [15] on top of SAC for a fair comparison. This is following the value expansions baseline implemented in MBPO [13].

Training Details: For LOOP-SAC we use SAC [5] as the underlying off-policy RL algorithm. Both the policy network (the parameterized actor) and the Q-function are parameterized by (256, 256) MLP with ReLU activations. The output of the policy network is a tanh squashed Gaussian. We use Adam to optimize both the policy and the Q-network with a learning rate of 3e-4. The temperature for SAC is learned to match a predefined target entropy. The replay buffer has a size of 1e6 and we use a batch size of 256. The target networks are updated with polyak averaging. Dynamics model related hyperparameters are listed in Table 2 and ARC related hyperparameters are in Table 3.

Hyperparameter	**Value**
Model Update frequency (K_m)	250
Ensemble Size (K)	5
Network Architecture	(200,200,200,200)
Model Learning rate	0.001

Table 2: Dynamics Model Hyperparameters

C.3 Offline RL

Baselines: We reimplement the CRR baseline in Pytorch. For PLAS, we use the original implementation [6]. Note that LOOP requires terminal Q-functions which estimate the cumulative value of future rewards. Some offline RL methods such as CQL will not be suitable to be combined with LOOP because CQL estimates a conservative lower-bound of the Q-function [31]. For MBOP [33], we report the results from their paper.

Training Details: For both LOOP-CRR and LOOP-PLAS we use the provided hyperparameters in the original papers. To optimize for the H-step lookahead objective given in Eqn. 3, we use ARC with

[2] https://github.com/vikashplus/mj_envs

[3] https://github.com/google-research/pddm/tree/master/pddm

[4] https://github.com/JannerM/mbpo

[5] https://github.com/openai/spinningup

[6] https://github.com/Wenxuan-Zhou/PLAS

Hyperparamater	Value
Planning Horizon (H)	3
Population Size (N)	100
Number of Particles (P)	4
Alpha (α)	0.1
Iterations (n_{ARC})	5
Mixture ratio (β)	0.05
Eta (η)	1

Table 3: Online RL: ARC Hyperparameters

1 iteration of Iterative importance sampling and $\beta = 1$ in the mixture prior. This is done to ensure that ARC trajectories are close to the actor trajectory distribution since the estimated Q-functions are only accurate within the data distribution. For each dataset, We perform an hyperparameter search over horizons h - [2,4,10], pessimism parameter β_{pess} - [0,0.5,1,5], exponential weighting temperature $1/\eta$ - [0.01,0.03,0.1,1,3,10] and noise standard deviation σ- [0.01,0.05,0.1]. We list the hyperparameters from the best experiments in Table 1.

Dataset-type	Environments	LOOP-CRR				LOOP-PLAS			
		h	$1/\eta$	β_{pess}	σ	h	$1/\eta$	β_{pess}	σ
random	hopper	2	10.0	0.5	0.4	4	3	0.5	0.4
	halfcheetah	2	0.01	5.0	0.4	2	1	0	0.01
	walker2d	4	3.0	0.0	0.05	10	10	0	0.05
medium	hopper	2	3.0	1.0	0.01	2	10	0.5	0.01
	halfcheetah	2	0.01	0.0	0.01	2	0.01	1.0	0.01
	walker2d	2	0.1	0.5	0.05	4	3.0	0	0.01
med-replay	hopper	4	0.03	0.5	0.01	2	3.0	1.0	0.01
	halfcheetah	2	1.0	0.5	0.01	2	0.03	5.0	0.01
	walker2d	4	1.0	0.5	0.1	4	0.01	0.5	0.01
med-expert	hopper	4	0.1	1.0	0.05	4	0.1	5.0	0.01
	halfcheetah	2	0.01	5.0	0.01	2	0.01	0.5	0.05
	walker2d	4	0.01	1.0	0.01	2	10.0	1.0	0.01

Table 4: Hyperparameters used in LOOP behavior policy during evaluation for Offline RL methods

C.4 Safe RL

Details on the Environments: For benchmarking safety environments we use the OpenAI safety gym environments [55]. We use a modified observation space for the agents where each agent observes its readings from velocimeter, magnetometer, and gyro sensors, LiDAR observations for the obstacles, and the goal location to a total of a 26-dimensional observation space. We also use an RC-car environment [56] in safe RL experiments shown in Figure 7. RC-car environment has a 6-dimensional observation space consisting of car's position and rate of change of its position. It's action space comprises of throttle and steer command.

Figure 7: Safety environments. Left to Right: PointGoal1, CarGoal1, Drift-v0

Baselines: We compare against CPO [57], LBPO [58], and PPO-lagrangian [59]. We use the original implementation for LBPO[7] and the safety benchmark [55] for CPO and PPO-lagrangian. All of the three baselines require a threshold to be set in order to optimize for safety. SafeLOOP optimizes for in-horizon safety whereas the baselines optimize for the infinite-step cumulative discounted return, so it becomes difficult to compare the methods directly. We design safeLOOP to optimize for 0 cost within the planning horizon and use the asymptotic safety cost reached by safeLOOP as the threshold for the baselines. We see that safeLOOP can reach average infinite horizon cost less than 10 which is lower than the threshold of 25 used in the official benchmark.

Training details: We use the safeARC algorithm presented in Algorithm 4 to solve the constrained optimization objective in Eqn. 10. The ARC parameters are the same as given in Table 3 with the Iterations(N) changed to 8 and Planning horizon(H) changed to 8. For OpenAI safety environments we use an action repeat of 5 across our method and the baselines.

[7] https://github.com/hari-sikchi/LBPO

D Additional Experiments

D.1 Online RL experiments for additional environments

Figure 8 shows the comparison of LOOP-SAC with baselines on additional tasks InvertedPendulum-v2, Swimmer, Hopper-v2 and TruncatedHumanoid-v2. LOOP-SAC is significantly more sample efficient than SAC as we observed in Figure 3. PETS-restricted has poor performance due to planning over a limited horizon. LOOP-SAC outperforms SAC-VE and is competitive to MBPO, except in Humanoid-v2 where MBPO outperforms. LOOP-SARSA has a poor performance across environments.

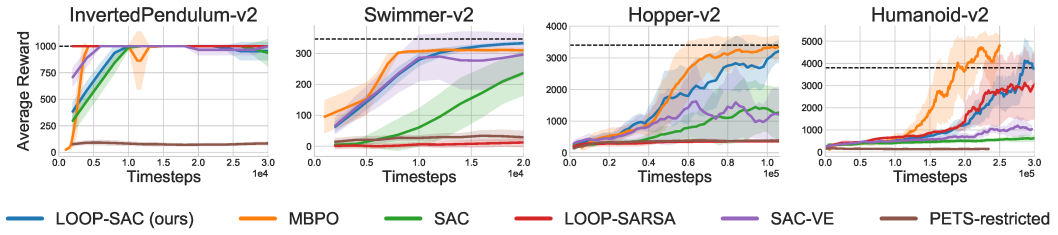

Figure 8: Comparisons of LOOP and the baselines for online RL for InvertedPendulum-v2, Swimmer, Hopper-v2 and TruncatedHumanoid-v2. LOOP-SAC is significantly more sample efficient than SAC. The dashed line indicates the performance of SAC at 1 million timesteps.

D.2 Comparison to modified POLO

In this section, we compare against POLO for Claw-v0 and HalfCheetah-v0 in Figure 9. The author's implementation of POLO is unavailable so we tried our best to implement it. To have a fair comparison with LOOP, we keep the hyperparameters as close to LOOP as possible and use a learned model; since the code for POLO is not available, we are unsure what hyperparameters were used in the original experiments. The performance of POLO is pretty low compared to LOOP, potentially due to the limited computation of CEM used for the value function update. Potentially the performance of POLO would be better with much larger computational resources than what we have available.

We would also like to highlight the difference in computational efficiency in LOOP and POLO. POLO requires an additional trajectory optimization procedure for value function computation, which is very computationally expensive. In contrast, LOOP learns a parameterized policy and value function to make the value function computation significantly faster. In addition, normally for online planning, we can warm start from the results from the previous time step ("amortization"); LOOP and PETS [17] take advantage of this optimization. In contrast, POLO cannot take advantage of this amortization during optimization for the value computation because we sample states IID from the replay buffer. Our implementation of POLO (after reasonable optimizations) takes ≈84 hours for 100k steps of HalfCheetah on a single NVIDIA 1080 GPU whereas LOOP takes ≈7 hours (12x less computation) while taking ≈1/5 the memory consumption of POLO.

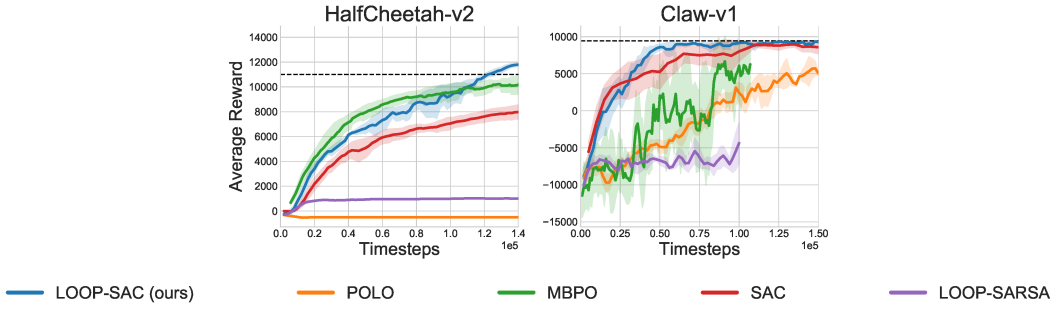

Figure 9: Comparisons of LOOP with modified-POLO for online RL for HalfCheetah-v2 and Claw-v1. POLO demonstrates poor performance which might be attributed to one of the reasons mentioned above.

D.3 Offline RL experiments for D4RL

Table 5 shows the performance of LOOP on four types of D4RL locomotion datasets. The random dataset is generated by a randomly initialized policy. The medium dataset is generated by executing a "medium quality" policy trained up to half of the final performance at convergence. The medium-replay dataset is the replay buffer of the medium quality policy. The medium-expert dataset is generated by a medium quality policy and a fully trained policy.

Dataset	Env	CRR	LOOP CRR	Improve%	PLAS	LOOP PLAS	Improve%	MBOP
random	hopper	10.40	10.68	2.7	10.35	10.71	3.5	**10.8**
	halfcheetah	4.23	7.55	78.5	26.05	**26.14**	0.3	6.3
	walker2d	1.94	2.04	5.2	0.89	2.83	218.0	**8.1**
medium	hopper	65.73	**85.83**	30.6	32.08	56.47	76.0	48.8
	halfcheetah	41.14	41.54	1.0	39.33	39.54	0.5	**44.6**
	walker2d	69.98	**79.18**	13.1	46.20	52.66	14.0	41.0
med-replay	hopper	27.69	29.08	5.0	29.29	**31.29**	6.8	12.4
	halfcheetah	42.29	42.84	1.3	43.96	**44.25**	0.7	42.3
	walker2d	19.84	27.30	37.6	35.59	**41.16**	15.7	9.7
med-expert	hopper	112.02	113.71	1.5	110.95	**114.32**	3.0	55.1
	halfcheetah	21.48	24.19	12.6	93.08	98.16	5.5	**105.9**
	walker2d	103.77	**105.76**	1.9	90.07	99.03	9.9	70.2

Table 5: Normalized scores for LOOP on the D4RL datasets comparing to the underlying offline RL algorithms and a baseline MBOP. LOOP improves the base algorithm across various types of datasets and environments.

D.4 Pessimism ablation for Offline RL

Table 6 shows an ablation of the pessimism term β_{pess} in Eqn. 9 as used in LOOP for Offline RL experiments. We note that the pessimistic term is not itself one of our contributions; this pessimistic term was used in previous works in model-based offline RL like [49, 50] which learn a policy given the data in an uncertainty penalized MDP. We observe that being pessimistic allows us to control incorrect extrapolation and obtain higher returns in most of the environments.

Dataset	Env	LOOP CRR ($\beta = 0$)	LOOP CRR ($\beta = \beta^*$)	β^*	LOOP PLAS ($\beta = 0$)	LOOP PLAS ($\beta = \beta^*$)	β^*
random	hopper	10.31	**10.68**	0.5	10.67	**10.71**	0.5
	halfcheetah	5.12	**7.55**	5.0	**26.14**	26.14	0.0
	walker2d	**2.04**	2.04	0.0	**2.83**	2.83	0.0
medium	hopper	78.56	**85.83**	1.0	54.97	**56.47**	0.5
	halfcheetah	**41.54**	41.54	0.0	38.01	**39.54**	1.0
	walker2d	75.21	**79.18**	0.5	**52.66**	52.66	0.0
med-replay	hopper	28.28	**29.08**	0.5	31.08	**31.29**	1.0
	halfcheetah	42.71	**42.84**	0.5	44.01	**44.25**	5.0
	walker2d	23.17	**27.30**	0.5	32.99	**41.16**	0.5
med-expert	hopper	104.57	**113.71**	1.0	98.87	**114.32**	5.0
	halfcheetah	23.84	**24.19**	5.0	94.19	**98.16**	0.5
	walker2d	104.57	**105.76**	1.0	97.87	**99.03**	1.0

Table 6: Normalized scores for LOOP on the D4RL datasets ablating the pessimism parameter.

D.5 Empirical analysis for ARC

In this section, we aim to verify how the ARC and its specfic hyperparameters affect the performance of LOOP for both the online RL and offline RL settings.

D.5.1 Ablation Study on Actor Regularized Control

In this experiment, we compare the performance of LOOP with ARC to a variant of LOOP which optimize Eqn. 3 without any constraint using CEM in the Online RL setting. CEM starts the optimization from the mean action sequence from the previous environment time step. It does not include actions proposed by the parameterized actor in the population. During training, we measure the actor-divergence defined to be the L_2 distance between the proposed action means of the parameterized actor and the CEM output.

The results are shown in Figure 10. The training process sometimes become unstable in the absence of ARC. We also observe that ARC empirically reduces actor-divergence during training.

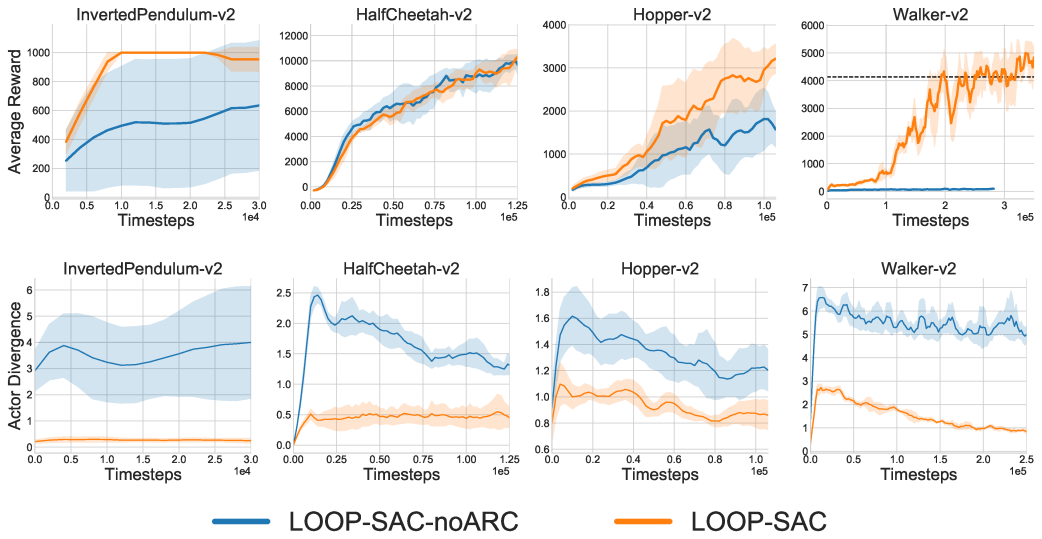

Figure 10: Top: We illustrate the effect of ARC on the performance for online-RL. Without ARC, the performance is worse and the training becomes unstable. Bottom: We illustrate that ARC effectively reduces the actor-divergence between the H-step lookahead policy and the parameterized actor.

D.5.2 ARC runtime

ARC runs at 14.3 Hz for the HalfCheetah-v1 environment with the hyperparameters specified in Table 3 on a machine with Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz and NVIDIA GeForce GTX 1080 Ti with a GPU memory consumption of 1500 MB.

D.5.3 Effect of β in ARC for Online RL

We see in Section 5.1 that ARC uses a mixture distribution of actor and 1-step shifted output from the previous timestep as the prior given by:

$$p^{\tau}_{prior} = \beta\pi_\theta + (1-\beta)\mathcal{N}(\mu_{t-1}, \sigma)$$

where β is the mixture coefficient.

In this experiment, we compare ARC with different parameters of beta for online RL. We observe empirically in Figure 11 that ARC with $\beta < 1$ is more suitable to online RL as it is less restrictive and allows for a greater improvement on the parametric actor.

D.5.4 Effect of β in ARC for in Offline RL

We use ARC with 1 importance sampling iteration for offline RL. In the following experiment, we compare ARC over 1 importance sampling iteration with $\beta = 1$ against ARC over 5 iterations with $\beta = 0.05$. The latter version utilizes the solution obtained by ARC in previous timestep which may potentially select trajectories that lead to out-of-distribution states with overestimated value. We see

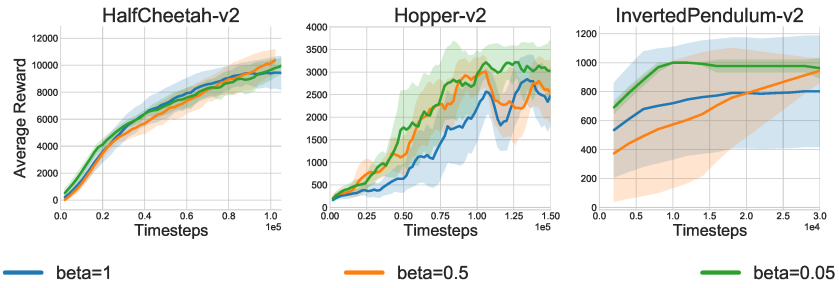

Figure 11: Effect of β in Online RL experiments. A high β constrains output actions to be close to actor and can restrict policy improvement.

in Table 7 that offline RL results for LOOP using ARC with $\beta = 0.05$ has much worse performance than ARC with $\beta = 1$ and also performs worse than the underlying offline RL method.

Dataset	Env	CRR	LOOP-CRR (β=0.05)	LOOP-CRR (β=1.0)	PLAS	LOOP-PLAS (β=0.05)	LOOP-PLAS (β=1.0)
random	hopper	10.40	7.50	10.68	10.35	7.66	10.71
	halfcheetah	4.23	2.32	7.55	26.05	5.21	26.14
	walker2d	1.94	2.43	2.04	0.89	1.24	2.83
medium	hopper	65.73	15.02	85.83	32.08	9.64	56.47
	halfcheetah	41.14	3.09	41.54	39.33	2.99	39.54
	walker2d	69.98	6.02	79.18	46.20	4.25	52.66
med-replay	hopper	27.69	8.78	29.08	29.29	6.8	31.29
	halfcheetah	42.29	3.10	42.84	43.96	4.68	44.25
	walker2d	19.84	6.02	27.30	35.59	6.89	41.16
med-expert	hopper	112.02	8.78	113.71	110.95	7.48	114.32
	halfcheetah	21.48	3.09	24.19	93.08	4.10	98.16
	walker2d	103.77	6.01	105.76	90.07	3.01	99.03

Table 7: Effect of β in offline-LOOP for the offline RL experiments. A low β can potentially select trajectories with overestimated returns.

D.6 Benefits of deploying H-step lookahead in Online RL

In LOOP we use a H-step lookahead policy for both exploration and evaluation. In this experiment, we run the Online RL experiments with LOOP only used for evaluation but not for exploration similar to the Offline RL experiments. This baseline is named SAC-evalLOOP. From Figure 12, LOOP-SAC outperforms both SAC-evalLOOP and SAC, which shows the benefits of H-step lookahead in LOOP during training-time deployment.

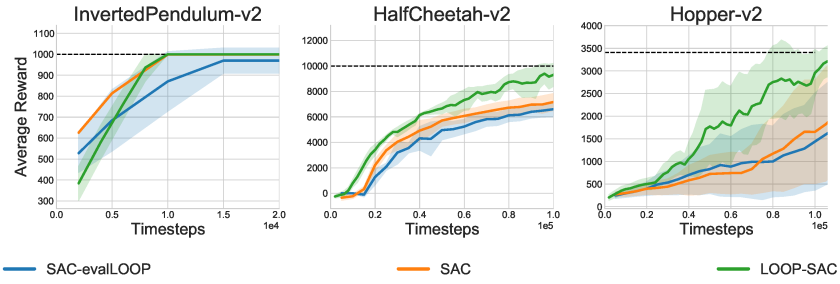

Figure 12: We compare the performance of LOOP-SAC to SAC-Online and SAC-Online-LOOPeval. LOOP-SAC outperforms both baselines and suggests that LOOP benefits from the H-step lookahead policy during training-time deployment.

D.7 Using Offline RL Algorithms with LOOP for Online RL

In Section 5.1, we mentioned that naively combining H-step lookahead policy with an off-policy algorithm will lead to the Actor Divergence issue. One potential solution we have considered besides ARC is to use an Offline RL algorithm as the underlying off-policy algorithm. Offline RL algorithm are designed to train a policy over a static dataset that is not collected by the parameterized actor which in principle should mitigate the instability issues of the value function learning caused by Actor Divergence. Note that in this case we are considering an Online RL *problem setting* with the help of Offline RL *algorithms*.

We investigate a combination of LOOP with Offline RL methods MOPO [49] and CRR [54]. We reimplement MOPO in PyTorch (originally in Tensorflow) for compatibility with other modules of LOOP. We also modify the dynamics model activations from Swish to ReLU. We use the same CRR implementation as the Offline RL experiments discussed above. We adapt MOPO and CRR to the Online RL setting by updating the policy and the value function for 20 gradient updates for each environment timestep. From Figure 13, LOOP-SAC has the most consistent performance across all the environments. LOOP-CRR and LOOP-MOPO work well in some cases but are significantly worse than LOOP-SAC in the others.

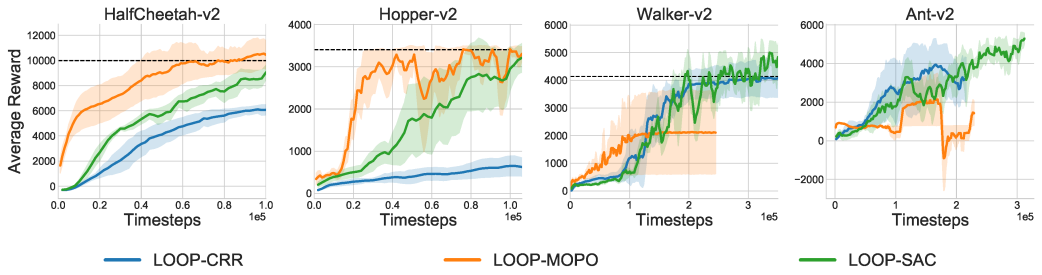

Figure 13: Using offline RL methods like CRR (model-free) or MOPO (model-based) with LOOP does not lead to consistently better performance.

Model Update frequency (K_m)	250
Ensemble Size	5
Network Architecture	MLP with 4 hidden layers of size 200
Model Horizon (H)	3
Model Learning rate	0.001
Policy update per environment step (R)	20
Replay Buffer Size	1e6
Gradient updates per timestep(R)	20
Pessimism parameter(λ)	1
Model rollout length	1

Table 8: LOOP-MOPO Hyperparameters