COPlanner: Plan to Roll Out Conservatively but to Explore Optimistically for Model-Based RL

Anonymous Author(s)
Affiliation
Address
`email`

Abstract: Dyna-style model-based reinforcement learning contains two phases: model rollouts to generate sample for policy learning and real environment exploration using current policy for dynamics model learning. However, due to the complex real-world environment, it is inevitable to learn an imperfect dynamics model with model prediction error, which can further mislead policy learning and result in sub-optimal solutions. In this paper, we propose `COPlanner`, a planning-driven framework for model-based methods to address the inaccurately learned dynamics model problem with conservative model rollouts and optimistic environment exploration. `COPlanner` leverages an uncertainty-aware policy-guided model predictive control (UP-MPC) component to plan for multi-step uncertainty estimation. This estimated uncertainty then serves as a penalty during model rollouts and as a bonus during real environment exploration respectively, to choose actions. Consequently, `COPlanner` can avoid model uncertain regions through conservative model rollouts, thereby alleviating the influence of model error. Simultaneously, it explores high-reward model uncertain regions to reduce model error actively through optimistic real environment exploration. `COPlanner` is a plug-and-play framework that can be applied to any dyna-style model-based methods. Experimental results on a series of proprioceptive and visual continuous control tasks demonstrate that both sample efficiency and asymptotic performance of strong model-based methods are significantly improved combined with `COPlanner`.

Keywords: Model-based RL, Model prediction error, Uncertainty-based Planning

1 Introduction

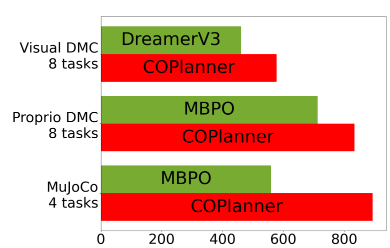

Figure 1: Mean performance of `COPlanner` compared with baselines across 3 diverse benchmarks.

Model-Based Reinforcement Learning (MBRL) has emerged as a promising approach to improve the sample efficiency of model-free RL methods. Most MBRL methods contain two phases that are alternated during training: 1) the first phase where the agent interacts with the real environment using a policy to obtain samples for dynamics model learning; 2) the second phase where the learned dynamics model rolls out to generate massive samples for updating the policy. Consequently, learning an accurate dynamics model is critical as the model-generated samples with high bias can mislead the policy learning [3, 33].

However, dynamics model errors are inevitable due to the complex real-world environment. Existing methods try to avoid model errors in two main ways. 1) Design different mechanisms such as filtering out error-prone samples to mitigate the influence of model errors after model rollouts [1, 35, 20, 34]. 2) Actively reduce model errors during real environment interaction through uncertainty-guided exploration [28, 24, 25, 18]. While both categories of methods have achieved advancements, each

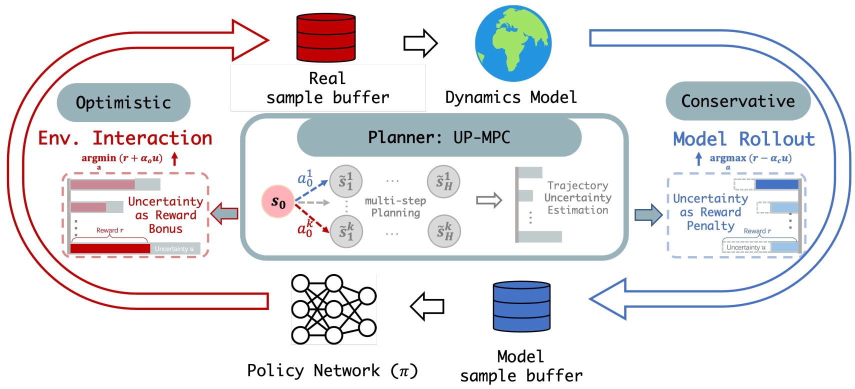

Figure 2: `COPlanner` Framework. The most essential part of `COPlanner` is the *Uncertainty-aware Policy-Guided MPC (UP-MPC) phase* in which we plan trajectories of length H, according to the learned dynamics model and learned policy network π, to select the action with highest trajectory reward. This UP-MPC phase is implemented differently for the two different purposes: *environment exploration* v.s. *dynamics model rollouts*. In environment exploration, trajectory reward has an uncertainty bonus term to encourage exploring uncertain regions in the environment. In dynamics model rollouts, trajectory reward, on the contrary, has an uncertainty penalty term to encourage policy learning on confident regions of the learned dynamics model.

comes with its own set of limitations. For the first category, although these approaches are shown to be empirically effective, they primarily concentrate on estimating uncertainty at the current step, often neglecting the long-term implications that present samples might have on model rollouts. Moreover, post-processing samples after model rollouts can compromise rollout efficiency as many model-generated samples are discarded or down-weighted. As for the second category, it is intrinsically challenging to achieve low model error and high long-term reward without sacrificing the sample efficiency by learning exploration policies.

To tackle the aforementioned limitations, we introduce a novel framework, `COPlanner`, which mitigates the model errors from two aspects: 1) avoid being misled by the existing model errors via conservative model rollouts, and 2) keep reducing the model error via optimistic environment exploration. The two aspects are achieved simultaneously by a novel uncertainty-aware multi-step planning method, which requires no extra exploration policy training nor additional samples, resulting in stable policy updates and high sample efficiency. `COPlanner` is structured around three core components: *the Planner*, *conservative model rollouts*, and *optimistic environment exploration*. In the Planner, we employ an *Uncertainty-aware Policy-guided Model Predictive Control* (UP-MPC) to forecast future trajectories in terms of selecting actions and to estimate the long-term uncertainty associated with each action. As shown in Figure 2, this long-term uncertainty serves as dual roles. In the model rollouts phase, the uncertainty acts as a penalty on the total planning trajectory, guiding the selection of conservative actions. Conversely, during the model learning phase, it serves as a bonus on the total planning trajectory, steering towards optimistic actions for environment exploration.

Compared to previous methods, `COPlanner` has the following advantages: **(a)** `COPlanner` has **higher exploration efficiency**, as it focuses on investigating high-reward uncertain regions to broaden the dynamics model, thereby preventing unnecessary excessive exploration of areas with low rewards. **(b)** `COPlanner` has **higher model-generated sample utilization rate.** Through planning for multi-step model uncertainty estimation, `COPlanner` can prevent model rolled out trajectories from falling into uncertain areas, thereby avoiding model errors before model rollouts and improving the utility of model generated samples. **(c)** `COPlanner` enjoys an **unified policy framework**. Unlike previous methods [25, 18] that require training two separate policies for different usage, `COPlanner` only requires training a single policy and we only change the way model-based planning is utilized, thus improving training efficiency and resolving potential policy distribution mismatches. **(d)** `COPlanner` ensures **undistracted policy optimization**. Notably, `COPlanner` diverges from existing approaches by not using long-term uncertainty as an intrinsic reward. Instead, the policy's objective remains focused on maximizing environmental rewards, thereby avoiding the introduction of spurious behaviors due to model uncertainty.

Summary of Contributions: **(1)** We introduce `COPlanner` framework which can mitigate the influence of model errors during model rollouts and explore the environment to actively reduce model errors simultaneously by leveraging our proposed uncertainty-aware policy-guided MPC. **(2)** `COPlanner` is a plug-and-play framework that can be applicable to any dyna-style MBRL method. **(3)** After being integrated with other MBRL baseline methods, `COPlanner` improves the sample efficiency of these baselines by nearly double. **(4)** Besides, `COPlanner` also significantly improves the performance on a suite of proprioceptive and visual control tasks compared with other MBRL baseline methods (16.9% on proprioceptive DMC, 59.7% on MuJoCo, and 23.9% on visual DMC).

2 Preliminaries

Model-based reinforcement learning. We consider a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \rho_0, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state space and action space respectively, $\mathcal{T}(s'|s,a)$ is the transition dynamics, ρ_0 is the initial state distribution, $r(s,a)$ is the reward function and γ is the discount factor. In model-based RL, the transition dynamics T in the real world is unknown, and we aim to construct a model $\hat{T}(s'|s,a)$ of transition dynamics and use it to find an optimal policy π which can maximize the expected sum of discounted rewards,

$$\pi = \underset{\pi}{\operatorname{argmax}} \, \mathbb{E}_{\substack{s_t \sim \hat{T}(\cdot|s_{t-1}, a_{t-1}) \\ a_t \sim \pi(a|s_t)}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \tag{1}$$

Model predictive control. Model predictive control (MPC) has a long history in robotics and control systems [5, 22]. MPC find the optimal action through trajectory optimization. Specifically, given the transition dynamics T in the real world, the agent obtains a local solution at each step t by estimating optimal actions over a finite horizon H (i.e., from t to $t+H$) and executing the first action a_t from the computed optimal sequence at time step t:

$$a_t = \underset{a_{t:t+H}}{\operatorname{argmax}} \, \mathbb{E} \left[\sum_{i=t}^{H} \gamma^i r(s_i, a_i) \right], s_i \sim T(\cdot|s_{i-1}, a_{i-1}), \tag{2}$$

where γ is typically set to 1. In model-based control methods, the transition dynamics T is simulated by the learned dynamics model $\hat{T}$ [2, 30, 11].

3 The `COPlanner` Framework

In this section, we will introduce `COPlanner` framework. `COPlanner` consists of three components: the Planner, conservative model rollouts, and optimistic environment exploration. Within the Planner, we propose using an Uncertainty-aware Policy-guided MPC to predict potential future trajectories when selecting different actions under the current state and estimate the long-term uncertainty associated with each action, which will be introduced in Sec 3.1. Depending on the phase, this long-term uncertainty is used to further guide the selection of conservative actions for policy learning or optimistic actions for environment exploration which will be introduced in Sec 3.2 and Sec 3.3.

3.1 "The Planner": Uncertainty-Aware Policy-Guided MPC

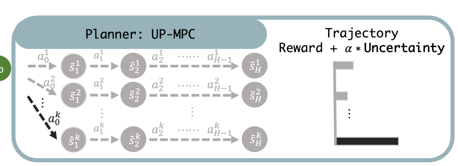

Figure 3: The Planner.

In this section, we present the core part of our proposed framework which is called Uncertainty-aware Policy-guided MPC (UP-MPC). Inspired by MPC, we apply the random shooting method [23] to introduce a long-term vision. Specifically, given the current state s_t, before each interaction with the model or real environment, we first generate an action candidate set containing K actions using the policy: $\boldsymbol{a_t} = \{a_t^{(1)}, a_t^{(2)}, ..., a_t^{(k)}\}$. Then, for each action candidate, we perform H_p-step planning and calculate the reward r, and model uncertainty u for each step. Finally, we select the action according to accumulated reward and model uncertainty, (to interact with the learned dynamics for the model rollouts or to interact with the environment for the model learning), as will be discussed in details in Sec 3.2 and Sec 3.3.

Incorporating model uncertainty is crucial for action selection to compensate for model error. As illustrated in Algorithm 1, we calculate the model uncertainty u through the model disagreement [21] method. Model disagreement is closely related to model learning and is currently the most common way to estimate model uncertainty in MBRL [35, 14, 20, 25, 34, 18]. We train a dynamics model ensemble $\hat{T}_\theta = \{\hat{T}_\theta^{(1)}, \hat{T}_\theta^{(2)}, ..., \hat{T}_\theta^{(n)}\}$ to predict the next state given the current state-action pair (s_t, a_t) as input. Utilizing the ensemble, we approximate the model uncertainty by calculating the variance over predicted states of the different ensemble members. This estimation closely represents the expected information gain [21]:

$$u(s_t, a_t) = \frac{1}{N-1} \sum_n (\hat{T}_\theta^{(n)}(s_t, a_t) - \mu')^2, \quad \mu' = \frac{1}{N} \sum_n \hat{T}_\theta^{(n)}(s_t, a_t). \tag{3}$$

See Figure 3 for the illustration of the process. The pseudocode for the Planner, i.e., the UP-MPC process, is summarized in Algorithm 1.

Algorithm 1 The Planner: **UP-MPC** $\left(\pi_\phi, s, \hat{T}_\theta, K, H, \alpha\right)$

Require: Policy π_ϕ, State s, learned dynamics model $\hat{T}_\theta$, number of candidates actions K, planning horizon H_p, optimistic/conservative parameter α

1: Initialize $R^{(k)} = 0$ for $k = 1, ..., K$, $s_0^{(k)} = s$ for $k = 1, ..., K$
2: **for** $k = 1$ to K **do**
3:     **for** $t = 0$ to $H_p - 1$ **do**
4:         Sample $a_t^{(k)} \sim \pi_\phi(\cdot|s_t^{(k)})$
5:         Rollout dynamics model $r_t^{(k)} = \hat{R}(\cdot|s_t^{(k)}, a_t^{(k)}),\ s_{t+1}^{(k)} \sim \hat{T}_\theta(\cdot|s_t^{(k)}, a_t^{(k)})$
6:         Compute model uncertainty $u_t^{(k)}$ according to Eq. 3
7:         $R^{(k)} = R^{(k)} + r_t^{(k)} + \alpha u_t^{(k)}$
8: Select $k^* = \arg\max_{k=1,...,K} R^{(k)}$
9: **return** $a_0^{(k^*)}$

Although in Algorithm 1 model uncertainty u is implemented through model disagreement, our proposed UP-MPC is a generic framework, any method for calculating intrinsic rewards to encourage exploration can be embedded into our framework for computing u. In Appendix D.6 we provide an ablation study of uncertainty estimation methods to further illustrate this point.

3.2 Conservative model rollouts

In model-based RL, due to the limited samples available for model learning, model prediction errors are inevitable. If a policy is trained using model-generated samples with a large error, these samples will not provide correct gradient and may mislead the policy update. Previous methods estimate the model uncertainty of each sample after generation and re-weight or discarded samples with high uncertainty. However, re-weighting samples based on uncertainty still leads to samples with high uncertainty participating in the policy learning process, while filtering requires manually setting an uncertainty threshold, and determining the optimal threshold is difficult. Discarding too many samples can result in inefficient rollouts.

We apply our Planner to plan for maximizing the future reward while minimizing the model uncertainty during model rollouts before executing the action. After calculating the reward and model uncertainty for the H_p-step trajectories of K action candidates (line 5 and 6 in Algorithm 1), we replace $\alpha = -\alpha_c$, for a positive $\alpha_c > 0$ at line 7 in Algorithm 1. Mathematically, we select the action according to Eq. 4 to interact with the model for model rollouts:

$$a = \text{argmax}_{a_t \in \boldsymbol{a_t}} \left[r(s_t, a_t) + \textstyle\sum_{i=1}^{H_p} r(\hat{s}_{t+i}, \pi(\hat{s}_{t+i})) - \alpha_c \textstyle\sum_{i=1}^{H_p} u(\hat{s}_{t+i}, \pi(\hat{s}_{t+i})) \right], \hat{s}_{t+i} \sim \hat{T}(\cdot|\hat{s}_{t+i-1}, a_{t+i-1}). \tag{4}$$

The negative $-\alpha_c$ is a coefficient that adds the model uncertainty as a penalty term to the trajectory total reward. By employing this approach, we can prevent model rollout trajectories from falling into model-uncertain regions while obtaining samples with higher rewards.

3.3 Optimistic environment exploration

In addition to model rollouts, another crucial part of MBRL is interacting with the real environment to obtain samples to improve the dynamics model. Since the main purpose of MBRL is to improve sample efficiency, we should acquire more meaningful samples for improving the dynamics model within a limited number of interactions. Therefore, unlike previous methods that merely aimed to thoroughly explore the environment to obtain a comprehensive model [28, 24, 25], we do not expect the dynamics model to learn all samples in the state space. This is because many low-reward samples do not contribute to policy improvement. Instead, we hope to obtain samples with both high rewards and high model uncertainty to sufficiently expand the model and reduce model uncertainty.

Similar to model rollouts, we also employ our Planner in the process of selecting actions when interacting with the environment. However, the difference lies in that we replace $\alpha = \alpha_o$, for a positive $\alpha_o > 0$ at line 7 in Algorithm 1. Mathematically, we choose the action with both high cumulative rewards and model uncertainty according to Eq. 5, which is a symmetric form of Eq. 4. α_o is a hyperparameter to balance the reward and exploration. Such an action can guide the trajectory towards regions with high rewards and model uncertainty in the real environment, thereby effectively expanding the learned dynamics model.

$$a = \mathrm{argmax}_{a_t \in \boldsymbol{a_t}} \left[r(s_t, a_t) + \sum_{i=1}^{H_p} r(\hat{s}_{t+i}, \pi(\hat{s}_{t+i})) + \alpha_o \sum_{i=1}^{H_p} u(\hat{s}_{t+i}, \pi(\hat{s}_{t+i})) \right], \hat{s}_{t+i} \sim \hat{T}(\cdot|\hat{s}_{t+i-1}, a_{t+i-1}) \tag{5}$$

In summary, by simultaneously using conservative model rollouts and optimistic environment exploration, `COPlanner` effectively alleviates the model error problem in MBRL. As we will show in Section 5, this is of great help in improving the sample efficiency and performance. The pseudocode of `COPlanner` is shown in Algorithm 2, and a more detailed figure is shown in Appendix A. Very importantly, `COPlanner` achieves both conservative model rollouts and optimistic environment exploration using a single policy. Different from prior exploration methods, the policy that `COPlanner` learns does not have to be an "exploration" policy which is inevitably suboptimal.

Algorithm 2 Main Algorithm: `COPlanner`

Require: Interaction epochs I, rollout horizon H_r, planning horizon H_p, number of candidates actions K, conservative rate α_c, optimistic rate α_o

```
Initialize policy π_φ, dynamics model T̂, real sample buffer D_e, model sample buffer D_m
for I epochs do
    while not Done do
        Select action a_t = UP-MPC(π_φ, s_t, T̂_θ, K, H_p, α_o)
        Execute in real environment, add (s_t, a_t, r_t, s_{t+1}) to D_e
        Train dynamics model T̂_θ with D_e
        for M model rollouts do
            Sample initial states from real sample buffer D_e
            for h = 0 to H_r do
                Select action â_{t+h} = UP-MPC(π_φ, ŝ_h, T̂_θ, K, H_p, −α_c)
                Rollout learned dynamics model and add to D_m
        Update current policy π_φ with D_m
```

4 Related work

Mitigating model error by improving rollout strategies. Prior methods primarily focus on using dynamics model ensembles [15, 2] to assess model uncertainty of samples after they were generated by the model, and then apply weighting techniques [1, 34], penalties [14, 35] or filtering [20, 32] to those high uncertainty samples to mitigate the influence of model error. These methods only quantify uncertainty after generating the samples and since their uncertainty metrics are based on the current step and are myopic, these metrics can not evaluate the potential influence of the current sample on future trajectories. Therefore, they fail to prevent the trajectories, which is generated through model rollout on the current policy, from entering high uncertainty regions, eventually leading to a failed policy update. Wu et al. [33] proposed Plan to Predict (P2P), which reverses the roles of the model and policy during model learning to learn an uncertainty-foreseeing model, aiming to avoid model uncertain regions during model rollouts. Combined with MPC, their method achieved promising results. However, their approach lacks effective exploration of the environment. Branched rollout [13] and bidirectional rollout [16] take advantage of small model errors in the early stages of rollouts and uses shorter rollout horizons to avoid model errors, but these approaches limit the planning capabilities of the learned dynamics model. Besides, different model learning objectives [27, 4, 31, 37] are designed to solve objective mismatch [17] in model-based RL and further mitigate model error during model rollouts.

Reducing model error by improving environment exploration. Another approach to mitigate model error is to expand the dynamics model by obtaining more diverse samples through exploration during interactions with the environment. However, previous methods mostly focused on pure exploration, i.e., how to make the dynamics model learn more comprehensively [28, 24, 25, 18, 12]. In complex environments, thoroughly exploring the entire environment is very sample-inefficient and not practical in real-world applications. Moreover, using pure exploration to expand the model may lead to the discovery of many low-reward samples (e.g., different ways an agent may fall in MuJoCo environment [29]), which are not very useful for policy learning. Mendonca et al. [18] proposed Latent Explorer Achiever (LEXA) which involves a explorer for exploring the environment and one achiever for solving diverse tasks based on collected samples, but the explorer and achiever may experience policy distribution shift under specific single-task settings, causing the achiever to potentially not converge to the optimal solution.

Mitigating model error from both sides. One most relevant work is Model-Ensemble Exploration and Exploitation (MEEE) [34] which simultaneously expands the dynamics model and reduces the impact of model error during model rollouts. During the rollout process, it uses uncertainty to weight the loss calculated for each sample to update the policy and the critic. Before interacting with the environment, they first generate k action candidates and then select the action with the highest sum of Q-value and one-step model uncertainty to execute. However, as we mentioned earlier, weighting samples cannot fundamentally prevent the impact of model errors on policy learning, and it may still mislead policy updates. Moreover, since the one-step prediction error of dynamics models is often small [20], relying only on the sum of Q-values and one-step model uncertainty may not effectively differentiate action candidates. As a result, samples collected during interactions with the environment might not efficiently expand the model.

5 Experiment

In this section, we combine `COPlanner` with strong MBRL baseline methods and conduct experiments on both proprioceptive control environments and visual control environments to demonstrate the effectiveness of our method. Due to space constraints, further discussions about the method and ablation studies can be found in Appendix D.

5.1 Experiment on proprioceptive control tasks

Baselines: In this section, we conduct experiments to demonstrate the effectiveness of `COPlanner` on proprioceptive control MBRL methods. We combine `COPlanner` with MBPO [13], the most classic method in proprioceptive control dyna-style MBRL, and we name the combined method as **COPlanner-MBPO**. The implementation details can be found in Appendix B. Consequently, **MBPO** naturally becomes one of our baselines. The other two baselines are **P2P-MPC** [33] and **MEEE** [34]. These two methods also aim to mitigate the impact of model errors in model-based RL. More details of P2P-MPC and MEEE can be found in Section 4. We also provide comparison with more proprioceptive control MBRL methods in Appendix D.1.

Environment and hyperparameter settings: We conduct experiments on 8 proprioceptive continuous control tasks of DeepMind Control (DMC) and 4 proprioceptive control tasks of MuJoCo.

MBPO trains an ensemble of 7 networks as the dynamics model while using the Soft Actor-Critic (SAC) as the policy network. In COPlanner-MBPO, we adopt the setting of MBPO and directly use the dynamics model ensemble to calculate model uncertainty for action selection in Policy-Guided MPC. For hyperparameter setting, we set optimistic rate α_o to be 1, conservative rate α_c to be 2 in most tasks. We set action candidate number K and planning horizon H_p equal to 5 in all tasks. The specific setting are shown in the Appendix C.1.

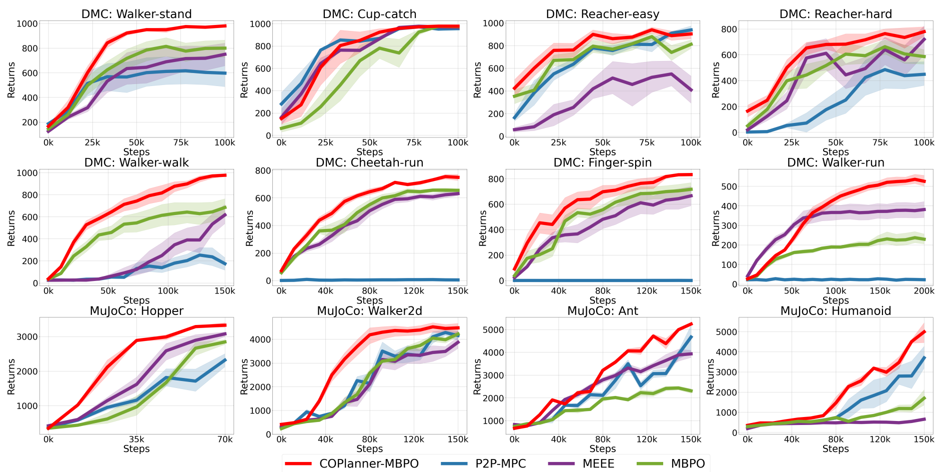

Figure 4: Experiment results of COPlanner-MBPO and other three baselines on proprioceptive control environments. The curves in the first eight figures originate from DM Control tasks, while those in the last four are from MuJoCo tasks. The results are averaged over 8 random seeds, and shaded regions correspond to the 95% confidence interval among seeds. During evaluation, for each seed of each method, we test for up to 1000 steps in the test environment and perform 10 evaluations to obtain an average value. The evaluation interval is every 1000 environment steps.

`COPlanner` **significantly improves the sample efficiency and performance of MBPO**: Through the results in Figure 4 we can find that both sample efficiency and performance of MBPO have a significant improvement after combining `COPlanner`. **(a) Sample efficiency**: In proprioceptive control DMC, the sample efficiency is improved by 40% on average compared to MBPO. For example, in the Walker-walk task, MBPO requires 100k steps for the performance to reach 700, while COPlanner-MBPO only needs approximately 60k steps. In more complex MuJoCo tasks, the improvement brought by `COPlanner` is even more significant. Compared to MBPO, the sample efficiency of COPlanner-MBPO has almost doubled. **(b) Performance**: From the performance perspective, as shown in Figure 1, the performance of MBPO has improved by 16.9% after combining `COPlanner`. Moreover, it is worth noting that our method successfully solves the Walker-run task, which MBPO fails to address, further demonstrating the effectiveness of our proposed framework. In MuJoCo tasks, the average performance at 150k environment steps has increased by 59.7%. Besides, COPlanner-MBPO also outperforms other two baselines.

5.2 Experiment on visual control tasks

Baselines: We conduct experiments to demonstrate the effectiveness of our proposed framework on visual control environments. We integrate our algorithm with DreamerV3 [10], the state-of-the-art Dyna-style model-based RL approach recently introduced for visual control. The implementation details can be found in Appendix B. We choose LEXA [18] as our another baseline. LEXA uses Plan2Explore [25] as intrinsic reward to explore the environment and learn a world model, then using this model to train a policy to solve diverse tasks such as goal achieving. Here we adopt LEXA on DreamerV3 to address continuous control tasks and name it as LEXA-DreamerV3. Since pure exploration base on Plan2Explore is sample inefficient for model learning when solving specific tasks, we use the real reward provided by environment as extrinsic reward and add it to intrinsic reward provided by Plan2Explore to train the explorer. We call this baseline LEXA-reward-DreamerV3.

We also provide comparison with more visual control MBRL methods including TDMPC [11] and PlaNet [8] in Appendix D.2.

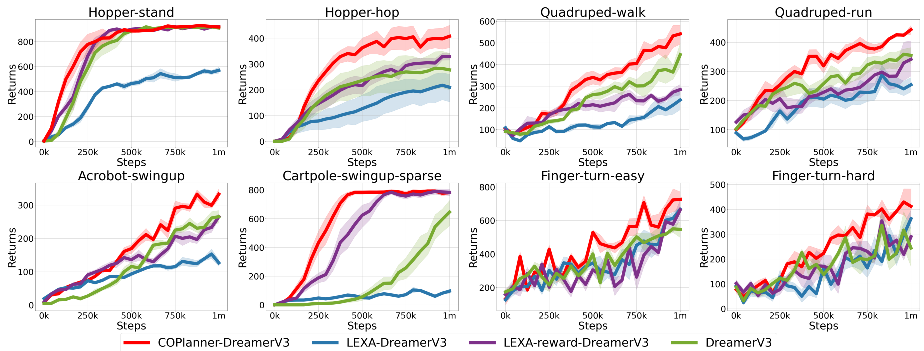

Figure 5: Experiment results of COPlanner-Dreamerv3 and other three baselines on pixel-input DMC. The results are averaged over 8 random seeds, and shaded regions correspond to the 95% confidence interval among seeds. During evaluation, for each seed of each method, we test for up to 1000 steps in the test environment and perform 10 evaluations to obtain an average value. The evaluation interval is every 1000 environment steps.

Environment and hyperparameter settings: We use 8 visual control tasks of DMC as our environment. In COPlanner-DreamerV3, we learn a latent one-step prediction dynamics model as Plan2Explore [25], the ensemble size is 8. We set action candidate number K and planning horizon H_p equal to 4 in all tasks. For optimistic rate α_o and conservative rate α_c, we set them to be 1 and 0.5, respectively. All other hyperparameters remain consistent with the original DreamerV3 paper.

`COPlanner` **significantly improves the sample efficiency and performance of DreamerV3**: From the experiment results in Figure 5, we observe that COPlanner-DreamerV3 improves the sample efficiency and performance significantly over DreamerV3. The sample efficiency of COPlanner-DreamerV3 is more than twice that of DreamerV3, and the performance is improved by 23.9%. Besides, LEXA-DreamerV3 has a low sample efficiency and do not perform well. This demonstrates the limitation of pure exploration when the goal is to solve specific tasks instead of learning a dynamics model applicable to a variety of tasks. After adding real reward as extrinsic reward for explorer learning, LEXA-reward-DreamerV3 delivers performance comparable to DreamerV3 in most environments. It outperforms DreamerV3 in Cartpole-swingup-sparse and Hopper-stand. However, its performance and sample efficiency are still worse than COPlanner-DreamerV3, further indicates the effectiveness of `COPlanner`.

6 Conclusion and discussion

We investigate how to effectively address the inaccurate learned dynamics model problem in MBRL. We propose `COPlanner`, a general framework that can be applied to any dyna-style MBRL method. `COPlanner` utilizes Uncertainty-aware Policy-Guided MPC phase to predict the cumulative uncertainty of future steps and symmetrically uses this uncertainty as a penalty or bonus to select actions for conservative model rollouts or optimistic environment exploration. In this way, `COPlanner` can avoid model uncertain areas before model rollouts to minimize the impact of model error, while also exploring high-reward model-uncertain areas in the environment to expand the model and reduce model error. Experiments on a range of continuous control tasks demonstrates the effectiveness of our method. One drawback of `COPlanner` is that MPC can lead to additional computational time and we provide a detailed computational time consumption in Appendix D.7. We can improve computational efficiency by parallelizing planning, which we leave for future work.

References

[1] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *Advances in neural*

information processing systems, 31, 2018.

[2] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in Neural Information Processing Systems*, 31, 2018.

[3] Marc Deisenroth and Carl 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, 2011.

[4] Benjamin Eysenbach, Alexander Khazatsky, Sergey Levine, and Russ R Salakhutdinov. Mismatched no more: Joint model-policy optimization for model-based rl. *Advances in Neural Information Processing Systems*, 35:23230–23243, 2022.

[5] Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[7] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[8] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[9] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[10] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[11] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. *arXiv preprint arXiv:2203.04955*, 2022.

[12] Edward S Hu, Richard Chang, Oleh Rybkin, and Dinesh Jayaraman. Planning goals for exploration. *arXiv preprint arXiv:2303.13002*, 2023.

[13] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

[14] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33:21810–21823, 2020.

[15] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *International Conference on Learning Representations*.

[16] Hang Lai, Jian Shen, Weinan Zhang, and Yong Yu. Bidirectional model-based policy optimization. In *International Conference on Machine Learning*, pages 5618–5627. PMLR, 2020.

[17] Nathan Lambert, Brandon Amos, Omry Yadan, and Roberto Calandra. Objective mismatch in model-based reinforcement learning. *arXiv preprint arXiv:2002.04523*, 2020.

[18] Russell Mendonca, Oleh Rybkin, Kostas Daniilidis, Danijar Hafner, and Deepak Pathak. Discovering and achieving goals via world models. *Advances in Neural Information Processing Systems*, 34:24379–24391, 2021.

[19] Andrew S Morgan, Daljeet Nandha, Georgia Chalvatzaki, Carlo D'Eramo, Aaron M Dollar, and Jan Peters. Model predictive actor-critic: Accelerating robot skill acquisition with deep reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6672–6678. IEEE, 2021.

[20] Feiyang Pan, Jia He, Dandan Tu, and Qing He. Trust the model when it is confident: Masked model-based actor-critic. *Advances in neural information processing systems*, 33:10537–10546, 2020.

[21] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International conference on machine learning*, pages 5062–5071. PMLR, 2019.

[22] S Joe Qin and Thomas A Badgwell. A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764, 2003.

[23] Anil V Rao. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135(1):497–528, 2009.

[24] Neale Ratzlaff, Qinxun Bai, Li Fuxin, and Wei Xu. Implicit generative modeling for efficient exploration. In *International Conference on Machine Learning*, pages 7985–7995. PMLR, 2020.

[25] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR, 2020.

[26] Younggyo Seo, Lili Chen, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. State entropy maximization with random encoders for efficient exploration. In *International Conference on Machine Learning*, pages 9443–9454. PMLR, 2021.

[27] Jian Shen, Han Zhao, Weinan Zhang, and Yong Yu. Model-based policy optimization with unsupervised model adaptation. *Advances in Neural Information Processing Systems*, 33: 2823–2834, 2020.

[28] Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International conference on machine learning*, pages 5779–5788. PMLR, 2019.

[29] 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, 2012.

[30] Tingwu Wang and Jimmy Ba. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*, 2019.

[31] Xiyao Wang, Wichayaporn Wongkamjan, Ruonan Jia, and Furong Huang. Live in the moment: Learning dynamics model adapted to evolving policy. In *International Conference on Machine Learning*. PMLR, 2023.

[32] Zhihai Wang, Jie Wang, Qi Zhou, Bin Li, and Houqiang Li. Sample-efficient reinforcement learning via conservative model-based actor-critic. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8612–8620, 2022.

[33] Zifan Wu, Chao Yu, Chen Chen, Jianye Hao, and Hankz Hankui Zhuo. Plan to predict: Learning an uncertainty-foreseeing model for model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 35:15849–15861, 2022.

[34] Yao Yao, Li Xiao, Zhicheng An, Wanpeng Zhang, and Dijun Luo. Sample efficient reinforcement learning via model-ensemble exploration and exploitation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4202–4208. IEEE, 2021.

[35] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33:14129–14142, 2020.

[36] Tianjun Zhang, Paria Rashidinejad, Jiantao Jiao, Yuandong Tian, Joseph E Gonzalez, and Stuart Russell. Made: Exploration via maximizing deviation from explored regions. *Advances in Neural Information Processing Systems*, 34:9663–9680, 2021.

[37] Ruijie Zheng, Xiyao Wang, Huazhe Xu, and Furong Huang. Is model ensemble necessary? model-based rl via a single model with lipschitz regularized value function. In *International Conference on Learning Representations*, 2023.

Appendix

A Detailed figure of COPlanner

We present a more detailed figure to illustrate our COPlanner framework. During environment exploration, we first choose an action using UP-MPC with multi-step uncertainty bonus, then interact with the real environment to obtain real samples for dynamics model learning. In dynamics model rollouts, at each rollout step, we select the actions using UP-MPC with multi-step uncertainty penalty to avoid model uncertain regions and interact with the learned dynamics model to get model-generated samples to update the policy.

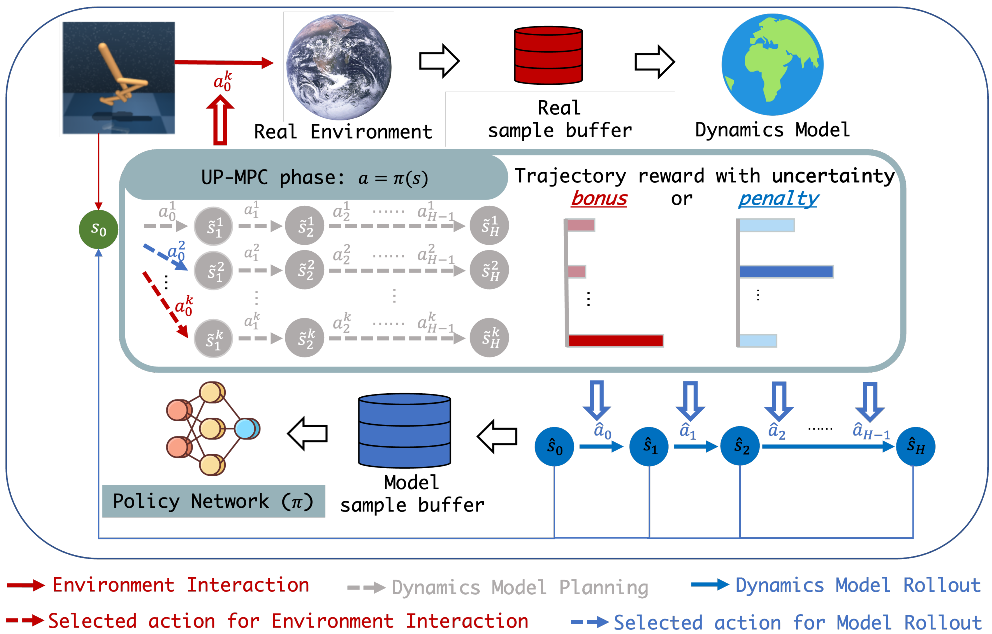

Figure 6: Figure illustration of COPlanner framework with more details.

B Implementation

COPlanner framework is versatile and applicable to any dyna-style MBRL algorithm. In this section, we are going to introduce the implementation of two algorithms we used for experiment in Section 5: COPlanner-MBPO for proprioceptive control and COPlanner-DreamerV3 for visual control.

B.1 COPlanner-MBPO

MBPO [13] trains an ensemble of probabilistic neural networks [2] as dynamics model. It utilises negative log-likelihood loss to update each network in the ensemble:

$$\mathcal{L}(\theta) = \sum_{n=1}^{N} [\mu_\theta^b(s_n, a_n) - s_{n+1}]^\top {\Sigma_\theta^b}^{-1}(s_n, a_n)[\mu_\theta^b(s_n, a_n) - s_{n+1}] + \log \det \Sigma_\theta^b(s_n, a_n) \tag{6}$$

For the policy component, MBPO adopts soft actor-critic [6]. We combine COPlanner with MBPO, the pseudocode is shown in Algorithm 3.

B.2 COPlanner-DreamerV3

DreamerV3 [10] is a dyna-style MBRL method that solves long-horizon tasks from visual inputs purely by latent imagination. Its world model consists of an image encoder, a Recurrent State-Space

Algorithm 3 COPlanner-MBPO

Require: interaction epochs I, rollout horizon H_r, planning horizon H_p, number of candidates actions K, conservative rate α_c, optimistic rate α_o

1: Initialize policy π_ϕ, dynamics model ensemble $\hat{T}_\theta = \{\hat{T}_\theta^1, ..., \hat{T}_\theta^i\}$, real sample buffer $\mathcal{D}_e$, model sample buffer $\mathcal{D}_m$
2: **for** I epochs **do**
3:   **for** t = 1 to T **do**
4:     *// Optimistic environment exploration*
5:     Select action with optimistic rate $a_t = \textbf{UP-MPC}\big(\pi_\phi, s_t, \hat{T}_\theta, K, H_p, \alpha_o\big)$
6:     Interact with the real environment with a_t, add real sample (s_t, a_t, r_t, s_{t+1}) to real sample buffer $\mathcal{D}_e$
7:     Train dynamics model $\hat{T}_\theta$ via Equation 6
8:     **for** M model rollouts **do**
9:       Sample initial rollout states from real sample buffer $\mathcal{D}_e$
10:       **for** h = 0 to $H_r - 1$ **do**
11:         *// Conservative model rollouts*
12:         $\hat{a}_h = \textbf{UP-MPC}\big(\pi_\phi, \hat{s}_h, \hat{T}_\theta, K, H_p, -\alpha_c\big)$ (Select action with conservative rate), rollout learned dynamics model and add to model sample buffer $\mathcal{D}_m$
13:     **for** G gradient updates **do**
14:       Update current policy π_ϕ using model-generated samples from model sample buffer $\mathcal{D}_m$

Model (RSSM) [8] to learn the dynamics, and predictors for the image, reward, and discount factor. The world model components are:

$$
\begin{aligned}
&\text{Recurrent model:} && h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1}) \\
&\text{Representation model:} && z_t \sim q_\phi(z_t | h_t, x_t) \\
&\text{Transition predictor:} && \hat{z}_t \sim p_\phi(\hat{z}_t | h_t) \\
&\text{Image predictor:} && \hat{x}_t \sim p_\phi(\hat{x}_t | h_t, z_t) \\
&\text{Reward predictor:} && \hat{r}_t \sim p_\phi(\hat{r}_t | h_t, z_t) \\
&\text{Discount predictor:} && \hat{\gamma}_t \sim p_\phi(\hat{\gamma}_t | h_t, z_t)
\end{aligned}
$$

where the recurrent model, the representation model, and the transition predictor are components of RSSM. The loss function for the world model learning is:

$$
\begin{aligned}
\mathcal{L}(\phi) = \mathbb{E}_{q_\phi(z_{1:T}|a_{1:T}, x_{1:T})}[\sum_{t=1}^{T} &(-\ln p_\phi(x_t|h_t, z_t) - \ln p_\phi(r_t|h_t, z_t) - \ln p_\phi(\gamma_t|h_t, z_t) \\
&+ \beta_1 max(1, \mathrm{KL}[sg(q_\phi(z_t|h_t, x_t))||p_\phi(z_t|h_t)]) \\
&+ \beta_2 max(1, \mathrm{KL}[q_\phi(z_t|h_t, x_t)||sg(p_\phi(z_t|h_t))]))],
\end{aligned} \tag{7}
$$

where sg means stop gradient. Besides, DreamerV3 also use actor-critic framework as their policy. In particular, they leverage a stochastic actor that chooses actions and a deterministic critic. The actor and critic are trained cooperatively. The actor goal is to output actions leading to states that maximize the critic output, while the critic aims to accurately estimate the sum of future rewards that the actor can achieve from each imagined state (or model rollout state). For more training details about DreamerV3, please refer to their original paper [10].

To estimate model uncertainty in COPlanner-DreamerV3, we train an ensemble of one-step predictive models $\hat{T}_\theta = \{\hat{T}_\theta^1, ..., \hat{T}_\theta^i\}$, each of these models takes a latent stochastic state z_t and action a_t as input and predicts the next latent deterministic recurrent states h_t. The ensemble is trained using MSE loss. During model rollouts, we use the world model to generate trajectories, and the one-step model ensemble to evaluate the uncertainty of sample at each rollout step. Here we provide the pseudocode of COPlanner-DreamerV3 in Algorithm 4.

Algorithm 4 COPlanner-DreamerV3

Require: Rollout horizon H_r, planning horizon H_p, number of candidates actions K, conservative rate α_c, optimistic rate α_o

Initialize real sample buffer $\mathcal{D}_e$ with S random seed episodes.
Initialize policy π_ψ, critic v_ξ, one-step model ensemble $\hat{T}_\theta = \{\hat{T}_\theta^1, ..., \hat{T}_\theta^i\}$, world model parameter ϕ
while not converged **do**
  for update step $c = 1..C$ **do**
    Draw $\mathcal{B}$ data sequences $\{(a_t, x_t, r_t)\}_{t=k}^{k+L} \sim \mathcal{D}_e$
    Compute a latent stochastic states $z_t \sim q_\phi(z_t | h_t, x_t)$
    Update world model parameter ϕ via Equation 7
    // Conservative model rollouts
    Imagine trajectories $\{(z_\tau, a_\tau)\}_{\tau=t}^{t+H_r}$ from each z_t with $a_\tau = \textbf{UP-MPC}\big(\pi_\psi, z_\tau, \hat{T}_\theta, K, H_p, -\alpha_c\big)$.
    Update v_ξ and π_ψ using imagined trajectories.
  for time step $t = 1..T$ **do**
    Compute $z_t \sim q_\phi(z_t | h_t, x_t)$
    // Optimistic environment exploration
    Select action $a_t = \textbf{UP-MPC}\big(\pi_\psi, z_t, \hat{T}_\theta, K, H_p, \alpha_o\big)$.
    Interact with the real environment and obatin $(x_t, a_t, r_t, x_t + 1)$
  Add experience to $\mathcal{D}_e \leftarrow \mathcal{D}_e \cup \{(x_t, a_t, r_t, x_t + 1)\}_{t=1}^{T}$

C Hyperparameters

In this section, we provide the specific parameters used in each task in our experiments.

Table 1: Hyperparameters of COPlanner-MBPO on proprioceptive control DMC.

Parameter	Value
Conservative rate α_c	2
Optimistic rate α_o	1
Action candidate number K	5
Planning horizon H_p	5
Real ratio	0.5 Reacher-xx 0.8 Finger-spin 0 Others
Rollout horizon H_r	[20, 150, 1, 1] Finger-spin [20, 150, 1, 4] Others

C.1 Proprioceptive control DMC and MuJoCo

We use COPlanner-MBPO in all proprioceptive control tasks. For the dynamics model ensemble, we adopted the same setup as MBPO [13] original paper, with an ensemble size of 7 and an elite number of 5, which means each time we select the best five out of seven neural networks for model rollouts. Each network in the ensemble is MLP with 4 hidden layers of size 200, using ReLU as the activation function. We train the dynamics model every 250 interaction steps with the environment. The actor and critic structures are both MLP with 4 hidden layers. In proprioceptive control DMC, the hidden layer size of actor and critic is 512, and updated 10 times each environment step, while in MuJoCo the hidden layer size is 256, and they are updated 20 times each environment step. The batch size for model training and policy training is both 256. The learning rate for model training is 1e-3, while the learning rate for policy training is 3e-4.

In MBPO, the authors use samples from both the real sample buffer and the model sample buffer to train the policy, and the ratio of the two is referred to as the real ratio. In addition, MBPO has a unique mechanism for the rollout horizon H_r, which linearly increases with the increase of environment epochs, with each environment epoch including 1000 environment steps. $[a, b, x, y]$ denotes a thresholded linear function, $i.e.$ at epoch e, rollout horizon is $h = \min(\max(x + \frac{e-a}{b-a}(y - x), x), y)$. The settings for conservative rate α_c, optimistic rate α_o, action candidate number K, planning horizon H_p and the above two parameters in different environments are provided in Table 1 and 2.

Table 2: Hyperparameters of COPlanner-MBPO on MuJoCo.

Parameter	Value
Conservative rate α_c	0.1 Hopper 2 Walker 0.5 Ant 1 Humanoid
Optimistic rate α_o	0.05 Hopper 1 Walker, Humanoid 0.1 Ant
Action candidate number K	5
Planning horizon H_p	5
Real ratio	0.05
Rollout horizon H_r	[20, 100, 1, 4] Hopper 1 Walker [20, 150, 1, 15] Ant [20, 300, 1, 15] Humanoid

C.2 Visual control DMC

In Visual control DMC, we use the COPlanner-DreamerV3 method. We keep all parameters consistent with the DreamerV3 original paper [10], except for our newly introduced conservative rate α_c, optimistic rate α_o, action candidate number K, and planning horizon H_p. In Table 3, we provide the specific settings of conservative rate α_c, optimistic rate α_o, action candidate number K, and planning horizon H_p for each task. It's worth noting that, although using a conservative rate of 0.5 can perform well, we find that for the two tasks in Quadruped, using a conservative rate of 2 yields the best sample efficiency and performance. For other parameters, please refer to the original DreamerV3 paper. For the one-step predictive model ensemble, we use a model ensemble with ensemble size of 8. Each network in the ensemble is MLP with 5 hidden layers of size 1024.

Table 3: Hyperparameters of COPlanner-DreamerV3 on visual control DMC. We keep all other hyperparameters consistent with the DreamerV3 original paper.

Parameter	Value
Conservative rate α_c	0.5
Optimistic rate α_o	1
Action candidate number K	4
Planning horizon H_p	4

D More experiments

D.1 Comparison with more proprioceptive control MBRL methods

In this section, we compared our approach with more proprioceptive control MBRL methods on MuJoCo tasks. In addition to the three baseline methods from Section 5.1, MBPO [13], P2P-MPC [33], and MEEE [34], we introduced two more baselines: PDML [31], a method that dynamically adjusts the weights of each sample in the real sample buffer to enhance the prediction accuracy of the learned dynamics model for the current policy, thereby significantly improving the performance of MBPO. And MoPAC [19], a method that also uses policy-guided MPC to reduce model bias. Unlike our approach, MoPAC's policy-guided MPC is solely used for multi-step prediction during rollout based on total reward to select actions. It does not incorporate a measure of model uncertainty, and therefore, cannot achieve the optimistic exploration and conservative rollouts of COPlanner. The experiment results are shown in Table 4.

As can be seen from Table 4, our method still holds a significant advantage, achieving the best performance in three tasks (Hopper, Walker2d, and Ant). In Humanoid task, it is only surpassed by PDML but is substantially better than the other methods. It's worth mentioning that our approach is orthogonal to PDML, and they can be combined. We believe that by integrating `COPlanner` with PDML, the performance can be further enhanced.

Table 4: Comparison of different MBRL methods on proprioceptive control MuJoCo tasks. Performance is averaged over 8 random seeds.

	Hopper (70k)	Walker2d (150k)	Ant (150k)	Humanoid (150k)
Ours	**3325.6 ± 153.7**	**4402.8 ± 376.5**	**5142.3 ± 138.3**	4994.3 ± 449.4
PDML	3274.2 ± 224.1	4378.5 ± 248.9	4992.5 ± 365.1	**5396.7 ± 391.3**
MBPO	2844.6 ± 158.0	4221.1 ± 281.1	2311.1 ± 252.5	1706.0 ± 976.3
P2P-MPC	2316.8 ± 459.9	4151.7 ± 516.9	4681.7 ± 591.6	3706.1 ± 1360.4
MEEE	3076.4 ± 165.3	3873.8 ± 549.6	3932.8 ± 352.7	654.2 ± 94.7
MoPAC	3174.2 ± 233.8	2893.6 ± 472.6	4382.5 ± 301.7	1084.6 ± 573.2

D.2 Comparison with more visual control MBRL methods

In this section, we conducted comparisons with more MBRL methods that use latent dynamics models for visual control on 8 tasks from visual DMC. In addition to DreamerV3 [10] and LEVA [18], we introduced two more baselines. The first is TDMPC [11]. TDMPC learns a task-oriented latent dynamics model and uses this model for planning. During the planning process, TDMPC also learns a policy to sample a small number of actions, thereby accelerating MPC. The second is PlaNet [8]. PlaNet uses the RSSM latent model, which is the same as the Dreamer series [7, 9, 10], and directly uses this model to perform MPC in the latent space to select actions. The experiment results are shown in Table 5. From the results, it is evident that our method has a significant advantage over all the baselines.

D.3 Experiments combined with DreamerV2

We also combine `COPlanner` with DreamerV2 [9] for experimentation, with the results shown in Figure 7. After integrating with DreamerV2, our method also achieves a significant improvement in both sample efficiency and performance.

D.4 Model error and rollout uncertainty analysis

In this section, we will investigate the impact of `COPlanner` on model learning and model rollouts. We provide the curves of how model prediction error and rollout uncertainty change as the environment step increases in Figure 8. We conduct experiments on two proprioceptive control DMC tasks

Table 5: Performance comparison of different MBRL methods on visual DMC tasks at 1 million environment steps.

	Hopper-stand	Hopper-hop	Quadruped-walk	Quadruped-run
Ours	**916.2** $\pm$ **19.0**	**406.6** $\pm$ **105.6**	**541.8** $\pm$ **113.6**	**443.4** $\pm$ **39.3**
DreamerV3	908.3 $\pm$ 21.7	277.9 $\pm$ 163.6	445.3 $\pm$ 129.8	354.9 $\pm$ 69.0
LEXA	569.6 $\pm$ 56.2	209.1 $\pm$ 126.3	235.3 $\pm$ 117.7	254.5 $\pm$ 61.0
TDMPC	821.6 $\pm$70.8	189.2 $\pm$ 19.7	427.8 $\pm$ 50.2	393.8 $\pm$ 40.9
PlaNet (5m)	5.96	0.37	238.90	280.45

	Acrobot-swingup	Cartpole-swingup-sparse	Finger-turn-easy	Finger-turn-hard
Ours	**332.8** $\pm$ **37.0**	**781.8** $\pm$ **24.5**	**724.9** $\pm$ **126.3**	**414.2** $\pm$ **166.6**
DreamerV3	264.8 $\pm$ 44.8	647.0 $\pm$ 193.1	545.8 $\pm$ 108.8	243.9 $\pm$ 180.9
LEXA	126.5 $\pm$ 25.0	95.3 $\pm$ 35.5	666.6 $\pm$ 36.6	362.5 $\pm$ 79.5
TDMPC	227.5 $\pm$ 16.9	668.3 $\pm$ 49.1	703.8 $\pm$ 65.2	402.7 $\pm$ 112.6
PlaNet (5m)	3.21	0.64	451.22	312.55

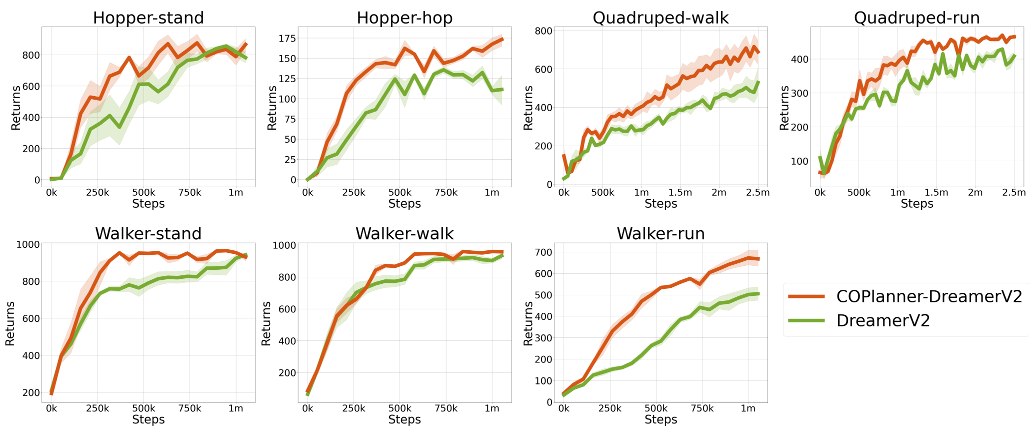

Figure 7: Experiment results of COPlanner-DreamerV2 on 7 visual control DMC tasks. The results are averaged over 4 random seeds, and shaded regions correspond to the 95% confidence interval among seeds. During evaluation, for each seed of each method, we test for up to 1000 steps in the test environment and perform 10 evaluations to obtain an average value. The evaluation interval is every 1000 environment steps.

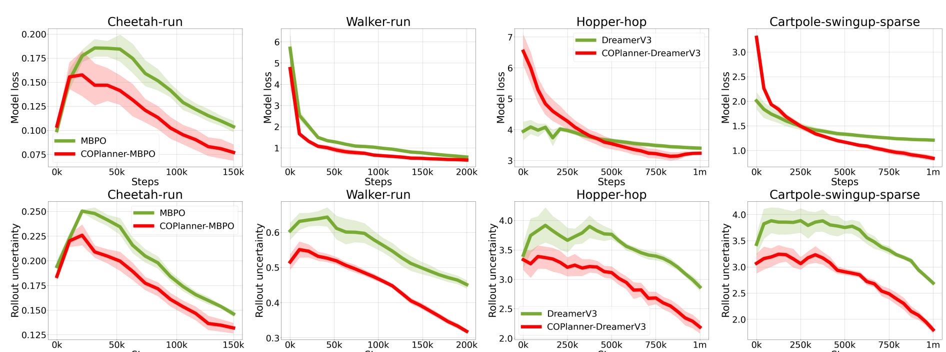

Figure 8: Model learning loss and rollout uncertainty curves for `COPlanner` and two other model-based RL baselines. The left four are proprioceptive control DMC tasks, and the right four are visual control DMC tasks.

(Cheetah-run and Walker-run) and two visual control DMC tasks (Hopper-hop and Cartpole-swingup-sparse).

(1) In proprioceptive control DMC, we use the MSE loss between model prediction and ground truth next state to evaluate model prediction error during training, while in visual control DMC, we use the KL divergence between the latent dynamics prediction and the next stochastic representation to compute latent model prediction error. We observe that after integrating `COPlanner` in proprioceptive control tasks, the model prediction error is significantly reduced. In more complex visual control tasks, due to obtaining more diverse samples through exploration in the early stages of training, the model prediction error of `COPlanner` is higher than the baseline (DreamerV3). However, as training progresses, the model prediction error rapidly decreases, becoming significantly lower than the model prediction error of DreamerV3. This allows the model to fully learn from the diverse samples, leading to an improvement in policy performance. **(2)** For the evaluation of rollouts uncertainty, we calculate the model disagreement for each sample in the model-generated replay buffer used for policy training using dynamics model ensemble. We find that `COPlanner` significantly reduces rollout uncertainty due to conservative rollouts, suggesting that the impact of model errors on policy learning is minimized. This experiment further demonstrates that the success of `COPlanner` is attributed to both optimistic exploration and conservative rollouts.

D.5 Hyperparameter study

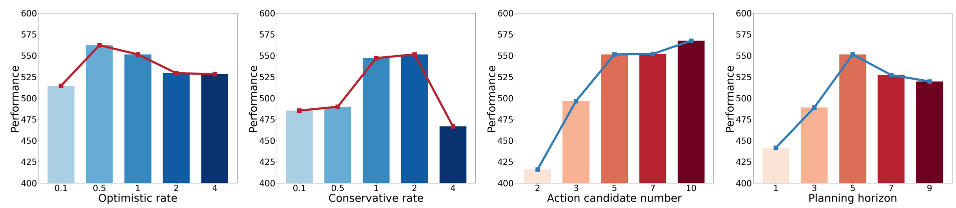

Figure 9: Ablation studies of `COPlanner`'s different hyperparameters. Experiments are conducted using COPlanner-MBPO on Walker-run tasks of proprioceptive control DMC. The results are averaged over 4 random seeds. From left to right, the results are for different parameters of optimistic rate α_o, conservative rate α_c, action candidate number K, and planning horizon H_p.

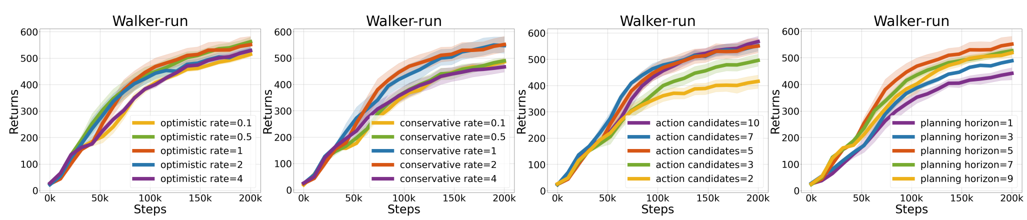

Figure 10: Performance curves of `COPlanner`'s hyperparameter study. Experiments are conducted using COPlanner-MBPO on Walker-run tasks of proprioceptive control DMC. The results are averaged over 4 random seeds.

In this section, we conducted hyperparameter studies to investigate the impact of different hyperparameters on `COPlanner`. We performed experiments on the Walker-run task of proprioceptive control DMC using COPlanner-MBPO. The original hyperparameter settings are: optimistic rate α_o is 1, conservative rate α_c is 2, action candidate number K is 5, and planning horizon H_p is 5. When conducting ablation experiments for each hyperparameter, other parameters remain unchanged. The results are shown together in Figure 9 and more detailed curves are given in Figure 10.

Optimistic rate α_o: we observe that the best α_o lies between 0.5 to 1. When the α_o is too large, `COPlanner` tends to excessively explore high uncertainty areas while neglecting rewards, leading to a decrease in sample efficiency and performance. On the other hand, when the α_o is too small, `COPlanner` fails to achieve the desired exploration effect.

Conservative rate α_c: the optimal range for the α_c is between 1 and 2. A too large α_c may lead to overly conservative selection of low-reward actions, while a too small α_c would be unable to make model rollouts avoid model uncertain areas.

Action candidate number K: we find that K has a significant impact on sample efficiency and performance. When K is set to 2, the improvement of `COPlanner` over MBPO in terms of performance and sample efficiency is relatively limited. This is reasonable because if there are only a few action candidates, our selection space is very limited, and even with the use of uncertainty bonus and penalty to select actions, there may not be much difference. When K increases to more than 5, the effect of `COPlanner` becomes very stable, and more candidates do not bring noticeable improvements in performance and sample efficiency.

Planning horizon H_p: when H_p is 1, we find that `COPlanner`'s improvement on performance and sample efficiency is relatively limited. This also confirms what we mentioned in Section 1: only considering the current step while ignoring the long-term uncertainty impact cannot completely avoid model errors, as samples with low current model uncertainty might still lead to future rollout trajectories falling into model uncertain regions. As the planning horizon gradually increases, performance and sample efficiency also rise. When the planning horizon is too long (H_p equals to 7 or 9), it is possible that due to the bottleneck of the model planning capability, most action candidates' corresponding trajectories fall into model uncertain areas, leading to a slight decline in performance and sample efficiency.

D.6 Ablation study of model uncertainty estimation methods

We conduct an ablation study on the Hopper-hop task in visual control DMC to evaluate different uncertainty estimation methods. We adopt two methods, RE3 [26] and MADE [36], which are used to estimate intrinsic rewards in pixel input, to replace the disagreement in calculating $u(s, a)$ in Equation 4 and 5. The results are shown in Figure 11. We find that the performance achieved using these two methods is similar to that of disagreement. This demonstrate that using disagreement to calculate uncertainty is not the primary reason for the observed performance improvement.

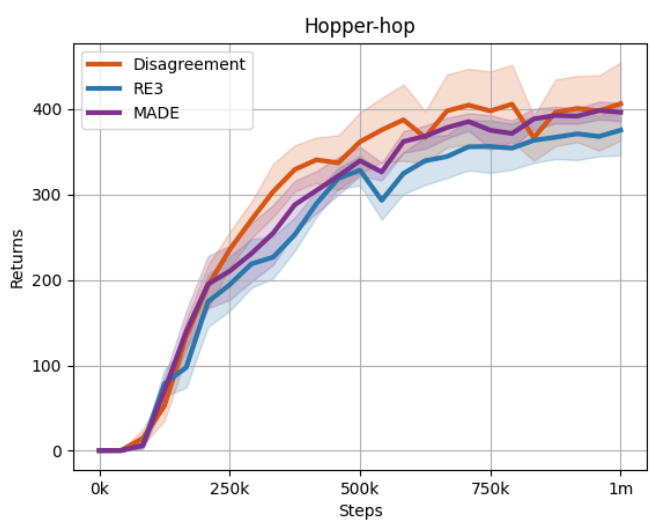

Figure 11: Ablation study of different uncertainty estimation methods.

D.7 Computational time consumption of `COPlanner`

We provide a comparison of the computational time consumption between the baseline methods and `COPlanner` across different domains in Table 6. All timings are reported using a single NVIDIA 2080ti GPU.

Table 6: Average time consumption (h).

	MuJoCo	Proprioceptive DMC	Visual DMC
COPlanner-MBPO	41.2	11.3	N/A
MBPO	33.78	10.6	N/A
COPlanner-DreamerV3	N/A	N/A	17.9
DreamerV3	N/A	N/A	13.1

D.8 Ablation study

In this section, we aim to investigate the impact of different components within `COPlanner` on the sample efficiency and performance. We conduct experiments on two proprioceptive control DMC tasks (Walker-stand and Walker-run) using MBPO as baseline and two visual control DMC tasks (Hopper-hop and Cartpole-swingup-sparse) with DreamerV3 as baseline. The results are demonstrated in Figure 12. Due to page limitations, we provide the ablation study on various hyperparameters of `COPlanner` in Appendix D.5 and the ablation study of uncertainty estimation methods in Appendix D.6.

From this ablation study, we can see that effectively combining optimistic exploration and conservative rollouts is necessary to achieve the best results. We find that when only using optimistic exploration (COPlanner w. Explore only), the sample efficiency and performance in all tasks are significantly improved, which highlights the importance of expanding the model. When only using conservative rollouts (COPlanner w. Rollout only), there is some improvement in sample efficiency and performance but to a lesser extent. In more complex visual control tasks, only using conservative rollouts may lead to over-conservatism, resulting in an inability to learn an effective policy in sparse reward environments (as observed with a broken seed in Cartpole-swingup-sparse) or a decrease in sample efficiency during the early stages of learning (Hopper-hop). This is reasonable because conservative rollouts may avoid high uncertainty and high reward areas to ensure the stability of policy updates. Moreover, without efficiently expanding the model, it is challenging to find better solutions using only conservative rollouts in complex visual control tasks. Experimental results show that both optimistic exploration and conservative rollouts are crucial, and using either one individually can lead to an improvement in performance. When combining the two (as in COPlanner), we can achieve the best results, further demonstrating the effectiveness of our method.

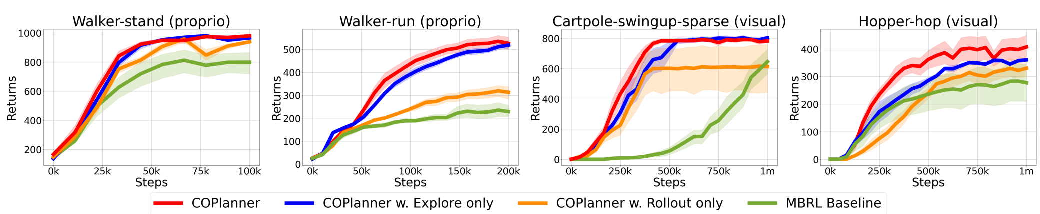

Figure 12: Ablation studies of optimistic exploration and conservative rollouts on different tasks using different mbrl baselines. In the first two proprioceptive control tasks we use MBPO as baseline. For the last two visual control tasks we employ DreamerV3. The results are averaged over 8 random seeds. We can observe that the best results are achieved when combining optimistic exploration and conservative rollouts. The benefit is more pronounced in more-challenging visual tasks.