A KL-REGULARIZATION FRAMEWORK FOR LEARNING TO PLAN WITH ADAPTIVE PRIORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Effective exploration remains a central challenge in model-based reinforcement learning (MBRL), particularly in high-dimensional continuous control tasks where sample efficiency is crucial. A prominent line of recent work leverages learned policies as proposal distributions for Model-Predictive Path Integral (MPPI) planning. Initial approaches update the sampling policy independently of the planner distribution, typically maximizing a learned value function with deterministic policy gradient and entropy regularization. However, because the states encountered during training depend on the MPPI planner, aligning the sampling policy with the planner improves the accuracy of value estimation and long-term performance. To this end, recent methods update the sampling policy by minimizing KL divergence to the planner distribution or by introducing planner-guided regularization into the policy update. In this work, we unify these MPPI-based reinforcement learning methods under a single framework by introducing Policy Optimization-Model Predictive Control (PO-MPC), a family of KL-regularized MBRL methods that integrate the planner's action distribution as a prior in policy optimization. By aligning the learned policy with the planner's behavior, PO-MPC allows more flexibility in the policy updates to trade off Return maximization and KL divergence minimization. We clarify how prior approaches emerge as special cases of this family, and we explore previously unstudied variations. Our experiments show that these extended configurations yield significant performance improvements, advancing the state of the art in MPPI-based RL.

1 INTRODUCTION

Recent approaches to planning-enhanced MBRL such as TD-MPC (Hansen et al., 2022) have shown that effective planning can significantly improve performance in MBRL by refining a learned policy through trajectory optimization. In these methods, a learned policy and its associated (action) value function are used for trajectory sampling and evaluation in a planning process (i.e. **sampling policy** and **bootstrap value function**). Then, the sampling policy is updated off-policy, relying on promising transitions provided by planning. This paradigm ensures that the planning policy continuously benefits from improvements in the learned sampling policy and bootstrap action value function, which supply increasingly promising samples and accurate evaluations to the planner.

A key limitation emerges when trajectories are evaluated under a bootstrap value function conditioned on states and actions unlikely to be visited by the planner. This distribution mismatch between the sampling and planning policies leads to unreliable bootstrap estimates and poor value function learning, especially for short horizons. Recent work addresses this by aligning the sampling policy with the planner via reverse KL minimization (Wang et al., 2025), but is hindered by its reliance on partially outdated planning samples, which introduce variance into policy updates.

Despite differing formulations, emerging MPPI-based methods implicitly follow the same principle for interacting with the environment and updating the policy, revealing a growing but fragmented landscape. This motivates a unifying framework that clarifies commonalities, organizes design choices, and enables systematic extensions to push forward the state of the art.

The main contribution of this work is Policy Optimization–Model Predictive Control (PO-MPC), a general MBRL framework for MPPI-based approaches. PO-MPC builds on the TD-MPC2 world model by casting the sampling policy learning step as an instance of KL-regularized RL, where the

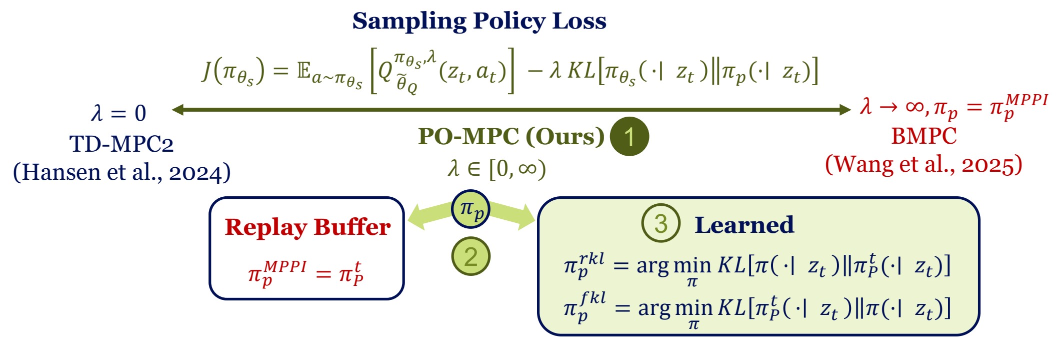
Sampling Policy Loss
$J(\pi_{\theta_s}) = \mathbb{E}_{a\sim\pi_{\theta_s}}\left[Q^{\pi_{\theta_s},\lambda}_{\tilde{\theta}_Q}(z_t, a_t)\right] - \lambda\, KL[\pi_{\theta_s}(\cdot|\ z_t)\|\pi_p(\cdot|\ z_t)]$
$\lambda = 0$
TD-MPC2
(Hansen et al., 2024)
$\lambda \to \infty, \pi_p = \pi_p^{MPPI}$
BMPC
(Wang et al., 2025)
PO-MPC (Ours) 1
$\lambda \in [0, \infty)$
π_p
2
Replay Buffer
$\pi_p^{MPPI} = \pi_P^t$
3 Learned
$\pi_p^{rkl} = \arg\min_{\pi} KL[\pi(\cdot|\ z_t)\|\pi_P^t(\cdot|\ z_t)]$
$\pi_p^{fkl} = \arg\min_{\pi} KL[\pi_P^t(\cdot|\ z_t)\|\pi(\cdot|\ z_t)]$

Figure 1: Overview of the PO-MPC framework. **1)** Sampling-policy learning is formulated as a KL-regularized reinforcement-learning problem, controlled by the hyperparameter λ, where the learned sampling policy π_{θ_s} is regularized toward the action distribution computed by MPPI. **2)** Since querying the MPPI policy is computationally expensive, we either reuse previously stored samples π_P^t or, as proposed in this work, learn an approximation of it (i.e., π_p^{rkl} or π_p^{fkl}). **3)** Using different losses to learn this policy prior, as a proxy for the planner's policy, allows embedding distinct inductive properties into the resulting sampling policy π_{θ_s}.

learned sampling policy π_{θ_s} is regularized against an MPPI-induced prior π_p with strength determined by a hyperparameter λ. In particular, our formulation enables:

- **Novel configurations.** We explore new algorithmic variants by tuning the KL-regularization strength λ.
- **Intermediate prior.** We introduce a learned prior that shields π_{θ_s} from outdated planner samples stored in the replay buffer.
- **Flexible objectives for training the prior.** We demonstrate how alternative losses for training the MPPI-induced prior embed distinct properties in π_{θ_s}, yielding superior performance.

We validate PO-MPC on challenging high-dimensional continuous control benchmarks, showing substantial gains in both sample efficiency and final performance over state-of-the-art baselines. These results highlight that a principled unification of MPPI-based approaches not only clarifies their design space but also drives concrete improvements in practice.

2 Related Work

Model-based RL. Model-based reinforcement learning (MBRL) (Moerland et al., 2023) studies the combination of model and policy learning in sequential decision-making problems. On the one hand, a learned model offers both extra data (Sutton, 1991) and/or allows planning and obtaining more informed actions (Silver et al., 2017) or value estimates (Feinberg et al., 2018). Conversely, learning offers an (approximate) solution over the entire input space that generalizes to unvisited state-actions (Ackley & Littman, 1989), which is indispensable to overcome the curse of dimensionality (Poggio et al., 2017).

Planning and RL. Our work builds on advancements in planning-based (and model-based) reinforcement learning (MBRL), particularly methods that leverage online planning to guide policy learning. In many such approaches, like TD-MPC and subsequent works Hansen et al. (2022; 2024), a learned policy provides initial actions for a trajectory optimizer or planner, which then refines these actions using a learned model. The optimized trajectories subsequently provide data for policy and value function updates. However, the policy update often relies only on the single best actions or resulting trajectories from the planner, discarding potentially valuable information about the broader action distribution explored during planning. Alternatively, Zhou et al. (2024) proposes using diffusion generative models to create policy and dynamic model proposals, and use them to solve an MPC problem.

Other examples of RL enhanced planning include Silver et al. (2017); Wang et al. (2025), where a policy is learned by imitating a powerful planner (e.g., MCTS, MPPI). Other methods exploit other sources of demonstrations to bias RL policies towards more informed distributions (Bhaskar et al., 2024; Hu et al., 2023; Yin et al., 2022). While effective, these imitation or cloning approaches may constrain the learned policy to the planner's immediate behavioral vicinity, potentially limiting its ability to directly optimize the long-term task objective (action value function) beyond what the planner currently achieves. On the other end, recent planning algorithms make use of expert knowledge or pre-trained policies to better inform the planning action search, robustly adapting to changes in the reward/cost function (Trevisan & Alonso-Mora, 2024; Wang et al., 2024a). In contrast, PO-MPC differentiates itself by proposing to utilize the entire action distribution generated by the planner, not just sampled actions or trajectories, as a guiding prior for the RL algorithm to exploit synergies between RL policy synthesis and planning-based action improvement.

RL as probabilistic inference. The idea of using priors to guide exploration in RL has been considered in many forms, albeit largely in the model-free setting (Tirumala et al., 2022). Priors can be used to guide learning by creating a trust region to constrain the optimization procedure (Schulman et al., 2015; 2017; Wang et al., 2017; Abdolmaleki et al., 2018); as an expectation-maximization (EM) update (Peters et al., 2010; Toussaint & Storkey, 2006; Rawlik et al., 2013; Levine & Koltun, 2013; Abdolmaleki et al., 2018) or to constrain learning in the offline or batch-RL setting (Siegel et al., 2020; Wu et al., 2019; Jaques et al., 2019; Laroche et al., 2017; Wang et al., 2020; Peng et al., 2020). A fundamental idea behind these works is to consider RL as a form of probabilistic inference where the policy being learned can be viewed as a posterior distribution over a prior and an objective (typically the exponentiated action value or advantage function) as in Levine (2018). In this work, we leverage this idea to reuse the model-based planning policy to guide learning its own sampling policy.

3 PRELIMINARIES

We consider a discrete-time sequential decision-making problem over a horizon T, modeled as a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $p(s' \mid s, a)$ the transition probability (or deterministic mapping) from state s to s' under action a, $r(s, a)$ the immediate reward for taking action a in state s, and $\gamma \in [0, 1)$ the discount factor. A policy $\pi(a \mid s)$ defines a distribution over actions given the current state, and the objective is to find π maximizing the expected discounted return

$$J(\pi) = \mathbb{E}_{\substack{s_0 \sim \rho_0, a_t \sim \pi(\cdot|s_t), \\ s_{t+1} \sim p(\cdot|s_t, a_t)}} \Big[\sum_{t=0}^{T-1} \gamma^t \, r(s_t, a_t) \Big], \tag{1}$$

where ρ_0 is the initial state distribution.

Reinforcement Learning (RL). does not assume direct knowledge of p or r; instead, an RL agent collects trajectories $\tau = (s_0, a_0, s_1, a_1, \ldots)$ through interaction and uses methods such as policy gradients, actor–critic, or value-based updates to learn a parametric policy $\pi_\theta(a \mid s)$ that maximizes $J(\pi_\theta)$ via trial-and-error.

Model Predictive Control (MPC). assumes access to a (possibly learned) model $p(s_{t+1} \mid s_t, a_t)$ and cost $c(s, a) = -r(s, a)$. At each time step t, MPC solves a finite-horizon optimization

$$\min_{a_{t:t+H-1}} \mathbb{E}\Big[\sum_{k=0}^{H-1} c(s_{t+k}, a_{t+k}) \Big] \quad \text{s.t.} \quad s_{t+k+1} = p(s_{t+k}, a_{t+k}), \tag{2}$$

over horizon $H < T$, applies the first action a_t, and then "recedes the horizon" by re-solving at $t + 1$ with the updated state. This online re-planning allows MPC to correct for model errors and disturbances. Both RL and MPC are methods to solve sequential decision-making optimisation problems: RL hinges on *learning* a global policy from experience, while MPC focuses on *online optimization* using an explicit model. In the next section, we show how Model Predictive Path Integral (MPPI) planning unifies these perspectives and can be further improved by incorporating learned policy priors via RL.

Model Predictive Path Integral Control. is a sample-based approach to solving planning methods that makes use of the fact that optimal stochastic control problems can be solved with path integrals to iteratively refine the optimal action distribution. At each step, it samples trajectories under a stochastic control law, weights them by cumulative cost, and refines its control sequence—all without requiring gradients of either dynamics or cost. Let $c(s_t, a_t)$ be a running cost (or reward $r = -c$) and H a finite planning horizon. Denote a nominal open-loop control sequence by $\bar{a}_{0:H-1} = (\bar{a}_0, \dots, \bar{a}_{H-1})$.

In MPPI, we sample M noisy trajectories $a_t^{(i)} = \bar{a}_t + \epsilon_t^{(i)}, \quad \epsilon_t^{(i)} \sim \mathcal{N}(0, \sigma_t I)$, and simulate $s_{t+1}^{(i)} \sim p\big(s_{t+1} \mid s_t^{(i)}, a_t^{(i)}\big)$. Each trajectory τ_i has an associated cost

$$S(\tau_i) = \sum_{t=0}^{H-1} c\big(s_t^{(i)}, a_t^{(i)}\big). \tag{3}$$

After selecting the K-top performing samples, the MPPI update follows from a path-integral (desirability) transform:

$$w_i = \frac{\exp\big(-\frac{1}{\lambda} S(\tau_i)\big)}{\sum_{j=1}^{K} \exp\big(-\frac{1}{\lambda} S(\tau_j)\big)}, \quad \bar{a}_t \leftarrow \bar{a}_t + \sum_{i=1}^{K} w_i\, \epsilon_t^{(i)}, \quad \sigma_t = \sqrt{\frac{\sum_{i=1}^{K} w_i \left(\epsilon_t^{(i)}\right)^2}{\sum_{i=1}^{K} w^i}} \tag{4}$$

where $\lambda > 0$ is the temperature parameter, and controls how much the importance sampling scheme weights the optimal cost trajectory versus the others. After a fixed number of iterations, the planning procedure is terminated and a trajectory is sampled from the final return-normalized distribution over action sequences. Planning is done at each decision step and only the first action is executed to produce a feedback policy. To warm-start optimization and speed convergence, the mean control sequence is initialized with the 1-step shifted $\bar{a}_{init} = \bar{a}_{t+1}$ from the previous decision step. We will denote the resulting **planning policy** obtained after a fixed number of MPPI iterations by $\mathcal{N}(\bar{a}_0, \sigma_0 I)$ and π_{P} interchangeably.

MPPI-based Reinforcement Learning. Prior work in Model-based RL (Bhardwaj et al., 2021; Hansen et al., 2022) has successfully applied MPPI to high-dimensional control tasks (i.e. DeepMind Control Suite (Tassa et al., 2018), Humanoid Benchmark (Sferrazza et al., 2024)) by planning in a learned a model of the MDP $(\hat{\mathcal{S}}, \mathcal{A}, \hat{p}, \hat{r}, \gamma)$, that differs from the original by using a learned latent representation of the state space $z = h_{\theta_h}(s) \in \hat{\mathcal{S}}$, an approximate reward $\hat{r}(z, a) = r_{\theta_r}$ and transition dynamics $\hat{p} = p_{\theta_d}$ (Bhardwaj et al., 2021).

In MPPI, trajectories are usually sampled from a Gaussian policy often initialized with zero mean and pre-set maximum variance to cover the action space almost uniformly, which is updated through multiple iterations of MPPI. Recent work (Hansen et al., 2024; Wang et al., 2025) biases this sampling distribution, augmenting it with trajectory samples produced with a **learned sampling policy:** π_{θ_s}. Since planning is done over a finite horizon, the learned sampling policy is also used for learning a **bootstrap action value function** $Q_{\theta_Q}^{\pi_{\theta_s}}$ evaluated on the last state of every sampled trajectory, leading to the H-step estimate: $Q(z_0, a_{0:H}^{(i)}) = \sum_{t=0}^{H-1} \gamma^t r_{\theta_r}(z_t, a_t^{(i)}) + \gamma^H Q_{\theta_Q}^{\pi_{\theta_s}}(z_H, a_H^{(i)})$.

Note that, since samples come from two distributions that are initially distinct, one learned and another initialized with high variance to enhance exploration. Then, the trajectory distribution is bi-modal. MPPI approximates a softmax posterior of the bi-modal distribution modulated by the normalized exponential of the estimated H-step value function returns. This process is reminiscent of epsilon-greedy policies, where high-return actions are taken with high probability, leaving some probability mass for exploration.

4 METHOD

4.1 POLICY UPDATE VARIANTS IN MPPI-BASED RL.

Initially, MPPI-based RL methods typically learned a *sampling policy* independently of the planner's action distribution. However, it is still influenced by the planner since it is used to collect the transition data used to update the sampling policy and the action value function. This decoupling creates a distribution mismatch: the value function is trained under states and actions induced by the planner, but the policy update optimizes a different objective (e.g. deterministic policy gradients with entropy regularization Hansen et al. (2024)[1].). For short horizons H, where trajectory scoring is dominated by the terminal bootstrap $Q^{\pi_{\theta_s}}_{\theta_Q}(z_H, a_H)$, this mismatch amplifies error. If π_{θ_s} is not aligned with MPPI, the states that Q implicitly predicts will not be reliably visited, degrading its estimates (Wang et al., 2025). Recent work addresses this by pulling the policy toward the planner by directly cloning the planning distribution via reverse KL minimization $\mathrm{KL}(\pi_{\theta_s}(\cdot|z_t) \,\|\, \pi_P(\cdot|z_t))$ (Wang et al., 2025)[2].However, this approach still suffers from:

- **Fixed KL penalty**: cloning the planning policy may collapse the sampling policy towards a local minima prematurely.
- **High-variance targets**: even when alleviated through *lazy reanalyze* (Wang et al., 2025), cloning uses stale planner statistics stored in the replay buffer that mix many planner versions, effectively turning a unimodal MPPI posterior into a time-varying Gaussian mixture.

We propose then to unify prior approaches under a single perspective: sampling policy learning as KL-regularized RL toward a *planner-induced prior*. This view makes explicit how design choices (trade-off between action-value maximization and KL minimization, Planning policy representation) map to previous methods, establish a generalised framework, and expose new, previously unexplored configurations.

4.2 Policy Optimization - Model Predictive Control

Given these considerations, we propose PO-MPC: a MBRL generalizing RL framework based on MPPI. The general algorithm pseudocode for PO-MPC training is presented in Algorithm 1. Following TD-MPC2's world model, previous approaches share a learned neural network **sampling policy**, π_{θ_s}, and the **bootstrap action value function** $Q^{\pi_{\theta_s}}_{\theta_Q}$, which are respectively used for biasing trajectory sampling and estimating the return of the trajectory beyond the horizon [3]. However, they all differ in how the learned sampling policy is updated. KL-regularized RL is a field of study that trains a policy to maximize its action-value function while regularizing the policy by minimizing the reverse KL-divergence to a second policy prior π_p. This regularization effect is modulated through a hyperparameter λ. In the following, we explain the main features of PO-MPC, being summarized as: **1)** Learning the sampling policy via KL-regularized RL, **2)** using a learned intermediate prior to represent the planning policy, which **3)** can be trained through different losses.

Sampling policy learning via KL-regularized RL. Given a state encoder $z = h_{\theta_h}(s)$ and a policy prior π_p, KL-regularized Reinforcement Learning considers the following goal in our framework:

$$J(\pi_{\theta_s}) = \mathbb{E}_{\substack{s_0\sim\rho_0, a_t\sim\pi_{\theta_s}(\cdot|z_t)\\ s_{t+1}\sim p(\cdot|s_t,a_t)}}\Big[\sum_{t=0}^{T-1}\gamma^t\big[r(z_t,a_t) - \lambda\mathrm{KL}[\pi_{\theta_s}(\cdot\mid z_t)\,\|\,\pi_p(\cdot\mid z_t)]\big]\Big], \tag{5}$$

where KL represents the Kullback-Liebler (KL) divergence between the policy and a prior distribution. The overall goal is to approximate, through the learned policy $\pi_{\theta_s}(\cdot \mid z_t)$, the distribution of trajectories generated by the prior policy $\pi_p(\cdot \mid z_t)$ reweighted by their exponential expected return. This is especially useful when prior policies are known that are likely to come across high-return regions in the state space, thus providing a promising trust region to explore around. As detailed in (Levine, 2018) for uniform policy priors, the objective in Equation 5 turns into the following step-wise objective:

$$J(\pi) = \mathbb{E}_{a\sim\pi_{\theta_s}}[Q^{\pi_{\theta_s},\lambda}_{\tilde{\theta}_Q}(z_t,a_t)] - \lambda\mathrm{KL}[\pi_{\theta_s}(\cdot\mid z_t)\,\|\,\pi_p(\cdot\mid z_t)], \tag{6}$$

[1]Although Hansen et al. (2024) reports using SAC for updating the sampling policy, their public code omits the entropy term in action value function estimation.

[2]Although Wang et al. (2025) reports minimizing the forward KL divergence, their public code minimizes the reverse KL, which leads to notable performance differences as discussed in this paper.

[3]Details on the implementation of MPPI and training of the bootstrapping action-value function can be found in Appendix B.

where d^{π} is the normalized state frequency visitation under the policy π_{θ_s}, and $Q^{\pi,\lambda}_{\tilde{\theta}_Q}$ is the KL-regularized action value function, which accounts for the expected return and the reverse KL divergence between the learned and the prior policy accumulated until the end of the episode. Then, the recursive Bellman equation for $Q^{\pi,\lambda}_{\tilde{\theta}_Q}$ is:

$$Q^{\pi_{\theta_s},\lambda}_{\tilde{\theta}_Q}(z_t, a_t) = \mathbb{E}_{\substack{s_{t+1}\sim p(\cdot|s_t,a_t),\\ a\sim\pi_{\theta_s}(\cdot|z_{t+1})}}\left[r(z_t, a_t) + \gamma\left(Q^{\pi_{\theta_s},\lambda}_{\tilde{\theta}_Q}(z_{t+1}, a) - \lambda\log\left(\frac{\pi_{\theta_s}(a \mid z_{t+1})}{\pi_p(a \mid z_{t+1})}\right)\right)\right] \tag{7}$$

Note that λ controls how close to the prior policy we want the sampling policy to be, which is enforced through Equations 6 and 7.

In this work, we focus on learning the sampling policy π_{θ_s}, and using the planning policy π_P for obtaining an adaptive prior π_p. We will also consider the case where we will maximize an entropy regularized objective $J'(\pi) = J(\pi) + \alpha\mathcal{H}(\pi)$, a term often included in KL-regularized RL to enhance exploration, as seen in Tirumala et al. (2022). We point the reader to Appendix E for additional proof on KL-regularized policy evaluation and improvement.

Prior policy design Setting $\lambda = 0$ updates the policy exclusively through action value function maximization and entropy regularization, recovering the cost function of TD-MPC2 (Hansen et al., 2024). Meanwhile, maximizing only the reverse KL-divergence of the policy and the past planning policy distributions stored in the replay buffer (i.e. $\lambda = \infty$) recovers the BMPC cost function (Wang et al., 2025).

We remark that this latter use of the planning policy samples as the prior introduces variance in the policy updates. The planning policy statistics (mean and variance) sampled from the replay buffer depend on old, less trained versions of the sampling policy. Therefore, for a given state, all sampled planning distributions have different modes, unlike the unimodal distribution that would result from MPPI under the current sampling policy and bootstrap action value function. This challenge is already recognized in Wang et al. (2025), and partially alleviated by periodically updating a small subset of the planning statistics sampled from the replay buffer.

We propose further decreasing the variance in the policy update by introducing an intermediate policy, an **adaptive prior** π_{θ_p}, that approximates the planning policy π_P. The benefits of this choice are twofold: **1)** it shields the sampling policy updates from the variance introduced by old planning policy samples (see Appendix G), and **2)** It can be trained with losses beyond reverse KL divergence, providing flexibility in how the planning policy π_P is represented and, in turn, how the sampling policy is guided.

As in prior methods, we can train this adaptive prior by either minimizing the reverse KL-divergence:

$$J(\theta_p) = \mathbb{E}_{(s,\pi_P)\sim D}\Big[\mathrm{KL}[\pi_{\theta_p}(\cdot \mid z_t) \parallel \pi_P(\cdot \mid z_t)]\Big], \tag{8}$$

or, as a straightforward alternative, the forward KL divergence:

$$J(\theta_p) = \mathbb{E}_{(s,\pi_P)\sim D}\Big[\mathrm{KL}[\pi_P(\cdot \mid z_t) \parallel \pi_{\theta_p}(\cdot \mid z_t)]\Big]. \tag{9}$$

Note that this choice comes with no loss of generality when the adaptive prior results from minimizing 8 (see Appendix G). Exclusively minimizing the reverse KL divergence between the learned sampling policy and the adaptive prior policy still recovers the policy update from Wang et al. (2025) since both sampling and adaptive prior policies are unimodal Gaussian distributions, and minimizing the reverse KL divergence imitates the latter exactly. Also note that choosing a prior that minimizes the reverse KL-divergence (Equation 8) will bias the sampling policy towards distributions that match one of the modes of the planning policy distribution, accelerating convergence but hurting exploration. Meanwhile, choosing priors that minimize the forward KL-divergence (Equation 9) will bias the policy towards a Gaussian distribution that encompasses the support of all sampled planning distributions, thereby enhancing exploration but delaying convergence. Further details on how the adaptive prior policy is trained are included in Appendix B.

Method Summary. PO-MPC provides a common view over previous methods while addressing two core challenges of MPPI-based RL: policy/planner mismatch and high-variance in stored planning

Algorithm 1 PO-MPC (Main): Plan $\to$ Infer $\to$ Regularize

```
Inputs: world model $\mathcal{M}$, simulated world model $\tilde{\mathcal{M}}$, MPPI planner, sampler policy $\pi_{\theta_s}$, value $Q_{\theta_Q}$, buffer $\mathcal{D}$,
KL weight $\lambda$, (optional) entropy $\alpha$
1: for $t = 0, \dots$ do
2:     Plan (policy-as-prior):
           • $\bar{a}_{t:t+H}, \sigma_{t:t+H} \leftarrow \mathrm{MPPI}_{\tilde{\mathcal{M}}}(z_t|\pi_{\theta_s}, Q^{\pi_{\theta_s}}_{\theta_Q}, \bar{a}_{init})$
           • $a_t \sim \pi_P(\cdot \mid z_t) := \mathcal{N}(\bar{a}_t, \sigma_t^2 \mathrm{I})$; step env to get $(r_t, s_{t+1})$;
           • Push $(s_t, a_t, r_t, s_{t+1}, \bar{a}_t, \sigma_t)$ to $\mathcal{D}$
3:     Update model $\tilde{\mathcal{M}}$ and Distill (adaptive prior): sample $\mathcal{B} \subset \mathcal{D}$; update $\theta_p$.
4:     Regularize & Improve (RL):
           1. Update $Q_{\theta_Q}$ and with TD targets under $\mathcal{M}$ using $\pi_{\theta_s}$.
           2. Update $Q^{\lambda}_{\hat{\theta}_Q}$ and with KL regularized TD targets under $\mathcal{M}$ using $\pi_{\theta_s}$.
           3. Update $\pi_{\theta_s}$ by maximizing $\mathbb{E}_{s\sim\mathcal{B},\, a\sim\pi_{\theta_s}}\big[Q^{\lambda}_{\hat{\theta}_Q}(z,a)\big] - \lambda\, \mathrm{KL}\big(\pi_{\theta_s} \| \pi_{\theta_p}\big) + \alpha\, \mathcal{H}(\pi_{\theta_s})$ .
5: end for
```

samples. We do this by casting policy learning as KL-regularized RL toward a distilled, *adaptive* planner prior. Concretely, MPPI produces a planning policy, which we distill into π_{θ_p} (via reverse or forward KL) to remove replay-induced variance; we then update the sampling policy π_{θ_s} with the KL-regularized objective in Eqs. 6–7, balancing return maximization, proximity to the planner (through λ), and entropy for exploration. This Plan$\to$Infer$\to$Regularize loop aligns the value function's rollout distribution with both the learned policy and the planner, improving stability and sample efficiency. The framework subsumes prior methods as special cases (λ=0 recovers TD-MPC2; $\lambda\to\infty$ with reverse-KL distillation recovers Variant 3[4] of Wang et al. (2025)) while enabling principled choice between fast mode-seeking convergence and broader support-covering exploration.

5 Experiments

We evaluate different configurations of the proposed framework (PO-MPC) on 7 challenging and high-dimensional continuous control tasks from DeepMind Control Suite (Tassa et al., 2018) (Humanoid and Dog) and 14 tasks from HumanoidBench locomotion suite (Sferrazza et al., 2024). These tasks cover a diverse range of continuous control challenges, including sparse reward, locomotion with high-dimensional state and action space ($\mathcal{A} \in \mathbb{R}^{21}$, $\mathcal{A} \in \mathbb{R}^{38}$, and $\mathcal{A} \in \mathbb{R}^{61}$ respectively). Each experiment is run on a single NVIDIA A100 GPU, taking from 7h to 15h to train a policy for 1e6 time-steps. For reproducibility, our implementation is available at `https://anonymous.4open.science/r/pompc-71E7`.

Baselines. We empirically support the claims in this work by comparing design choices already taken under this framework in the literature, namely TD-MPC2 (Hansen et al., 2024) and BMPC (Wang et al., 2025).We also explore simple variations in this framework by studying the effect of different values of λ, the inclusion of the intermediate policy π_{θ_p}, and how it is trained. Table 2 provides an overview of the tested configurations, including published works. Since BMPC learns the value rather than the action-value function, setting $\lambda = \infty$ (minimizing only the KL-divergence in 6) recovers Variant 3 of Wang et al. (2025). This detail does not affect our policy update analysis. We evaluate each baseline with the updated hyperparameters from its repository. We evaluate PO-MPC under the same hyperparameters of TD-MPC2, with the exception of those related to PO-MPC (see Appendix A).

5.1 Results

The objective of this section is to test PO-MPC from three angles. First, we make an empirical study of the effects of prioritizing return maximization over KL divergence minimization by choosing different values for λ. Second, we verify that employing an intermediate policy prior does not hurt the performance of PO-MPC. Finally, we show an example of how different policy priors may serve to embed different properties in the sampling policy.

[4]Variant 3 inWang et al. (2025) is identical to BMPC except for the fact that it learns the bootstrap action value function $Q^{\pi_{\theta_s}}$ of the sampling policy instead of the value function $V^{\pi_{\theta_s}}$.

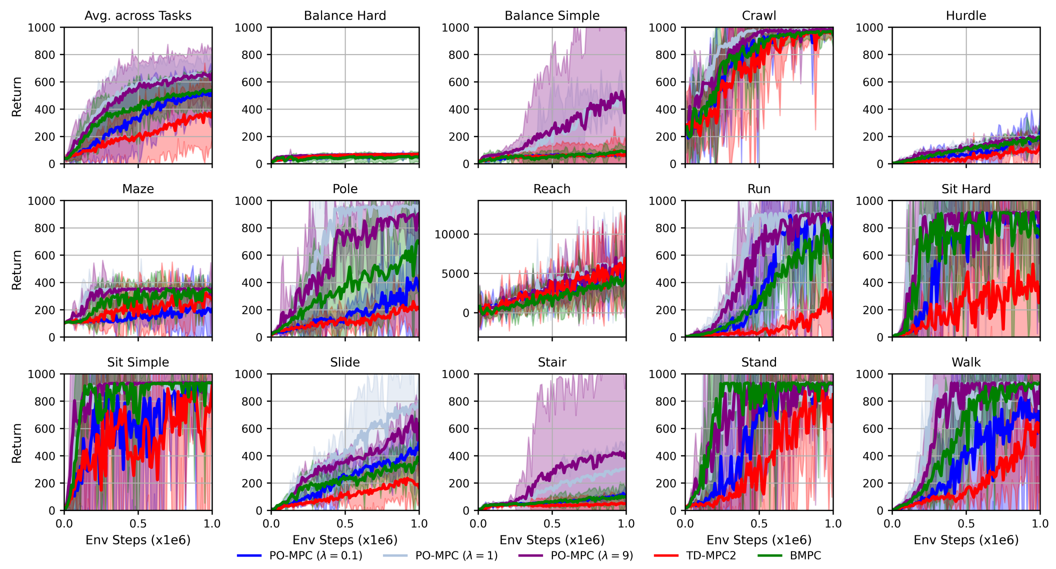

Figure 2: Performance comparison in 14 state-based high-dimensional control tasks from HumanoidBench (Sferrazza et al., 2024). Mean of 3 runs; shaded areas are 95% confidence intervals. In the top left, we visualize results averaged across all tasks except for *Reach* due to different range.

Table 1: Final performance across 7 high-dimensional control tasks from DMControl Suite (Tassa et al., 2018). Mean of 3 runs and 95% CI. Learning curves are reported in Appendix D

	Dog				Humanoid		
	Stand	Trot	Walk	Run	Stand	Walk	Run
TD-MPC2	978±6	738±488	957±9	611±76	915±33	910±34	480±60
BMPC	992±8	931±11	964±11	740±107	950±34	946±4	529±90
Ours (λ=0.1)	993±4	959±12	976±12	709±66	958±10	948±17	581±90
Ours (λ=1)	993±4	946±11	966±14	720±132	959±14	948±3	554±101
Ours (λ=9)	990±2	959±18	974±27	703±166	958±10	948±19	548±22

Trading off return and KL divergence optimization. The parameter λ regulates the trade-off between two competing objectives in the policy updates: maximizing episode returns and minimizing the KL divergence from the adaptive policy prior. Table 1 and Figure 2 show PO-MPC evaluations with a policy prior learned according to Equation 8, under different values of λ. Specifically, we consider $\lambda = 0.1, 1$, and 9.0, which correspond to approximate prioritization of KL divergence minimization of 10%, 50%, and 90%, respectively. Our results demonstrate that regulating the proximity between the sampling and planning policies matches the performance of the baselines in the lower-dimensional tasks from DMControl Suite while significantly boosting performance in the higher-dimensional tasks from HumanoidBench. In particular, in the latter tasks, intermediate values of λ never perform worse than the baselines and often clearly outperform them (i.e., Balance S., Crawl, Pole, Run, Slide, Stair, and Walk). Overall, when averaged across tasks, PO-MPC with intermediate λ values is on par in low-dimensional tasks (DMControl), but achieves clear superior results with respect to the state-of-the-art in higher-dimensional ones (HumanoidBench), especially when λ is carefully tuned.

Policy prior: Learned Intermediate policy vs. Planning replay data. Continuing our experiments in HumanoidBench, Figure 3 shows that, on average across tasks, using a learned intermediate policy instead of using the Planning policy samples from the replay buffer matches the performance of the latter and, in some cases, surpasses it. We hypothesize this is due to the reduction in variance that results from the intermediate policy prior being able to be approximated exactly by the sampling

Table 2: Method characteristics and empirical trends under the PO-MPC view. Arrows denote trends observed in our experiments; details in Figs. 2–4 and Table 1. Performance and Sample efficiency are taken w.r.t. TD-MPC2. Note that $\lambda \to \infty$ means only the KL divergence in Eq. 6 is optimized.

Method	**Uses planning policy prior**	**KL-reg. objective**	**Fwd/Rev KL** (Eq. 8, 9)	**Sample eff.**	**Final perf.**
TD-MPC2	✗($\lambda = 0$)	✗	–	baseline	baseline
BMPC	✓($\lambda \to \infty$)	✓ (π_P)	Fwd	$\uparrow$	$\uparrow$ / $\approx$
PO-MPC (Ours)	✓(λ var.)	✓ (π_{θ_p})	Fwd / Rev	$\uparrow$	$\uparrow\uparrow$

policy, instead of the ensemble of, partially outdated, unimodal Gaussian Planning policy samples from the replay buffer.

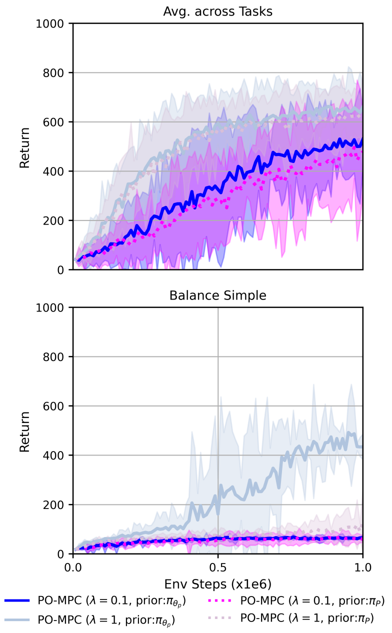

Figure 3: Effects of using a learned intermediate prior, π_{θ_p}, instead of the Planning samples, π_P, from the replay buffer. Mean of 3 runs; shaded areas are 95% CI. We report the average across tasks **(Top)** and in the Balance Simple task **(Bottom)**. See Appendix D for results on all tasks.

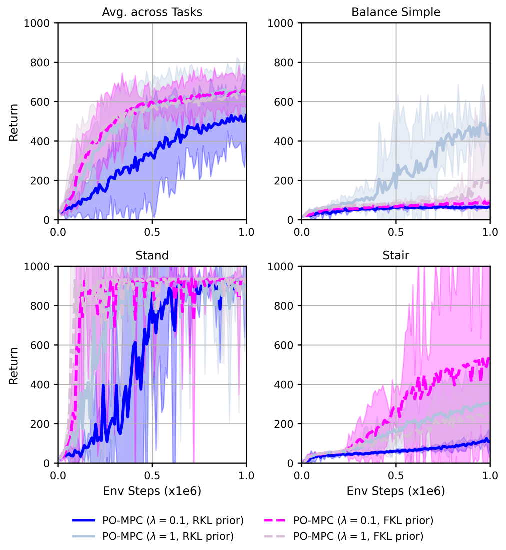

Figure 4: Effects of approximating the Planning policy with the intermediate prior through different cost functions. Mean of 3 runs; shaded areas are 95% CI. We report the average across tasks, and environments showing a clear effect of training with loss in Eq. 9 instead of Eq. 8. See Appendix D for results on all tasks.

Policy prior Training. Figure 4 exemplifies how depending on the environment, choosing an alternative policy prior will change the effect of our chosen value for λ, improving or deteriorating the performance. For example, choosing priors minimizing the forward KL-divergence (Equation 9) will bias the policy towards a Gaussian distribution that includes the support of all sampled planning distributions, instead of matching the most frequent mode in the batch. This enhances exploration but delays convergence. This is why it is beneficial in environments where exploration is key, converging to a more stable solution faster at low values of λ (i.e., in Stair); but detrimental in environments where deterministic behavior is crucial to obtain high rewards (i.e. Balance Simple).

6 Discussion and Conclusion

Summary of Findings Across 7 DMControl (Humanoid/Dog) and 14 HumanoidBench tasks, PO-MPC consistently improves over TD-MPC2 and is competitive with or exceeds BMPC. Figures 5–2 show that even modest KL regularization (e.g., $\lambda = 0.1$) yields sizable gains over TD-MPC2, with larger λ often dominating in high-dimensional settings. Replacing on-replay planner samples with a learned *adaptive prior* matches or surpasses cloning-from-replay (Fig. 3), suggesting reduced update variance and smoother training (see Appendix G). The choice of prior fitting objective is task-dependent: forward KL tends to help exploration-heavy tasks (e.g., Stair) at low λ, whereas reverse KL accelerates convergence on precision-dominated tasks (e.g., Balance Simple) (Fig. 4). These results support the main claims of the work: closing the loop so that planning informs policy updates (and vice-versa) yields guided exploration and better sample efficiency in MPPI-based RL.

Limitations. Tuning hyperparameter λ is essential for the performance of PO-MPC. As a rule of thumb, we keep it to $\lambda = 1$, to equally weight return maximization and KL minimization. However, its optimal value depends both on the complexity of the environment and the training of the policy prior. A similar approach might be taken as in SAC (Haarnoja et al., 2018), where the appropriate value of λ would be learned during training.

Also, information obtained during planning is not fully exploited. Many trajectories are simulated during planning that, although used for computing an action sequence, are not leveraged for learning the action value function, thus being computationally inefficient. Additionally, such trajectories are constrained to short horizons. The model loses accuracy at long horizons, which reduces the accuracy of the estimated scores for each sampled trajectory as well.

Finally, we assume both learned sampling policy and policy prior to be Gaussian distributions. This approximation is very restrictive since the Planning policy, which consists of a Gaussian prior reweighted by an exponential distribution of the trajectory costs, is not necessarily Gaussian.

Conclusion. This paper introduced *Policy Optimization – Model Predictive Control* (PO-MPC), a family of model-based reinforcement learning methods for continuous action spaces. In particular, PO-MPC extends MPPI-based RL by finding a common formulation that includes previously published approaches in the state-of-the-art, and exploits previously unexplored design choices. Our experiments show that PO-MPC leveraging these choices often learn faster and more stably than the other baselines, serving as a new state-of-the-art for model-based RL in continuous action spaces. Future work could focus on **1)** extending the distribution of the policies used to more expressive classes than Gaussian, **2)** automatically tuning the trade-off between Return maximization and KL minimization, and **3)** increasing the computational efficiency by leveraging the simulated transition data generated during planning for action value learning.

REFERENCES

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=S1ANxQW0b`.

David Ackley and Michael Littman. Generalization and scaling in reinforcement learning. *Advances in neural information processing systems*, 2, 1989.

Mohak Bhardwaj, Sanjiban Choudhury, and Byron Boots. Blending {mpc} & value function approximation for efficient reinforcement learning. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=RqCC_00Bg7V`.

Amisha Bhaskar, Zahiruddin Mahammad, Sachin R Jadhav, and Pratap Tokekar. Planrl: A motion planning and imitation learning framework to bootstrap reinforcement learning. *arXiv preprint arXiv:2408.04054*, 2024.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/jax-ml/jax`.

Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

Shane Flandermeyer. tdmpc2-jax: Jax/flax implementation of TD-MPC2. `https://github.com/ShaneFlandermeyer/tdmpc2-jax`, 2024a. Accessed: 2025-08-28.

Shane Flandermeyer. bmpc-jax: Jax/flax implementation of BMPC. `https://github.com/ShaneFlandermeyer/bmpc-jax`, 2024b. Accessed: 2025-08-28.

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*, pp. 1861–1870. Pmlr, 2018.

Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations (ICLR)*, 2024.

Nicklas A Hansen, Hao Su, and Xiaolong Wang. Temporal difference learning for model predictive control. *International Conference on Machine Learning*, pp. 8387–8406, 2022.

Hengyuan Hu, Suvir Mirchandani, and Dorsa Sadigh. Imitation bootstrapped reinforcement learning. *arXiv preprint arXiv:2311.02198*, 2023.

Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog, 2019.

Romain Laroche, Paul Trichelair, and Rémi Tachet des Combes. Safe policy improvement with baseline bootstrapping, 2017.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

Sergey Levine and Vladlen Koltun. Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems*, pp. 207–215, 2013.

Haotian Lin, Pengcheng Wang, Jeff Schneider, and Guanya Shi. Td-m(PC)^2: Improving temporal difference mpc through policy constraint. *arXiv preprint arXiv:2502.03550*, 2025.

Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage weighted regression: Simple and scalable off-policy reinforcement learning, 2020. URL `https://openreview.net/forum?id=H1gdF34FvS`.

Jan Peters, Katharina Mülling, and Yasemin Altün. Relative entropy policy search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pp. 1607–1612. AAAI Press, 2010.

Tomaso Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing*, 14(5):503–519, 2017.

Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference (extended abstract). In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pp. 3052–3056. AAAI Press, 2013. ISBN 9781577356332.

Julian Schrittwieser, Thomas K Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and offline reinforcement learning by planning with a learned model. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=HKtsGW-lNbw`.

John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Carmelo Sferrazza, Dun-Ming Huang, Xingyu Lin, Youngwoon Lee, and Pieter Abbeel. HumanoidBench: Simulated Humanoid Benchmark for Whole-Body Locomotion and Manipulation. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.061.

Noah Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. Keep doing what worked: Behavior modelling priors for offline reinforcement learning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rke7geHtwH`.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

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

Richard S. Sutton and Andrew G. Barto. *Generalized Policy Iteration*, chapter 4, pp. 76–80. MIT Press, Cambridge, MA, 2nd edition, 2018. Section 4.2.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. DeepMind Control Suite. Technical report, DeepMind, 2018.

Dhruva Tirumala, Alexandre Galashov, Hyeonwoo Noh, Leonard Hasenclever, Razvan Pascanu, Jonathan Schwarz, Guillaume Desjardins, Wojciech Marian Czarnecki, Arun Ahuja, Yee Whye Teh, and Nicolas Heess. Behavior priors for efficient reinforcement learning. *Journal of Machine Learning Research*, 23(221):1–68, 2022. URL `http://jmlr.org/papers/v23/20-1038.html`.

Marc Toussaint and Amos Storkey. Probabilistic inference for solving discrete and continuous state markov decision processes. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pp. 945–952, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933832. doi: 10.1145/1143844.1143963. URL `https://doi.org/10.1145/1143844.1143963`.

Elia Trevisan and Javier Alonso-Mora. Biased-mppi: Informing sampling-based model predictive control by fusing ancillary controllers. *IEEE Robotics and Automation Letters*, 2024.

Pengcheng Wang, Chenran Li, Catherine Weaver, Kenta Kawamoto, Masayoshi Tomizuka, Chen Tang, and Wei Zhan. Residual-mppi: Online policy customization for continuous control. *arXiv preprint arXiv:2407.00898*, 2024a.

Shengjie Wang, Shaohuai Liu, Weirui Ye, Jiacheng You, and Yang Gao. Efficientzero v2: Mastering discrete and continuous control with limited data. In *Forty-first International Conference on Machine Learning*, 2024b. URL `https://openreview.net/forum?id=LHGMXcr6zx`.

Yuhang Wang, Hanwei Guo, Sizhe Wang, Long Qian, and Xuguang Lan. Bootstrapped model predictive control. *arXiv preprint arXiv:2503.18871*, 2025.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. In *International Conference on Learning Representations*, 2017.

Ziyu Wang, Alexander Novikov, Konrad Zolna, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, and Nando de Freitas. Critic regularized regression, 2020.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning, 2019.

Zhao-Heng Yin, Weirui Ye, Qifeng Chen, and Yang Gao. Planning for sample efficient imitation learning. *Advances in Neural Information Processing Systems*, 35:2577–2589, 2022.

Guangyao Zhou, Sivaramakrishnan Swaminathan, Rajkumar Vasudeva Raju, J Swaroop Guntupalli, Wolfgang Lehrach, Joseph Ortiz, Antoine Dedieu, Miguel Lázaro-Gredilla, and Kevin Murphy. Diffusion model predictive control. *arXiv preprint arXiv:2410.05364*, 2024.

Zifeng Zhuang, Diyuan Shi, Runze Suo, Xiao He, Hongyin Zhang, Ting Wang, Shangke Lyu, and Donglin Wang. Tdmpbc: Self-imitative reinforcement learning for humanoid robot control. *arXiv preprint arXiv:2502.17322*, 2025.

A HYPERPARAMETERS

In table 3 we share the hyperparameters employed for both our method (PO-MPC) and the baseline TD-MPC. Both methods share all parameters except for the ones exclusive to PO-MPC.

Table 3: Hyperparameter configuration.

Hyperparameters	**Values**
General	
Num. steps	1 000 000
Replay buffer	1 000 000
Learning_rate	3e-4
Max. Gradient norm	20
Optimizer	Adam($\beta_1 = 0.9$,$\beta_2 = 0.999$)
World model	
Encoder dim.	256
Num. Encoder layers	2
Learning_rate	3e-4
Latent_dim	512
Dropout	0.01
Num. Value Nets	5
Num. bins	101
Symlog min,max	-10, 10
Simnorm dim	8
TD-MPC2	
Horizon	3
MPPI iterations	8
Population size	512
Policy prior samples	24
Num. elites	64
Min. plan std (σ_{min})	0.05
Max. plan std (σ_{max})	2
Temperature	1.0
Batch size (n_r)	256
Discount (γ)	0.99
Time discount (ρ)	0.5
Consistency coef.	20
Reward model coef.	0.1
Value function coef.	0.1
Entropy coef. (α)	1e-4
Target update coef. (τ)	0.01
PO-MPC	
Biased value function coef.	0.1
KL Reg. strength λ	{0.1, 1.0, 9.0}
Learned intermediate prior policy	{Yes, No}
Prior policy learning loss	{Fwd KL, Rev KL}
Reanalyzed batch (n_b^r)	20
Reanalyzed interval (k)	10

B Implementation details

In this appendix, we give a thorough explanation of the procedure followed to implement PO-MPC. For the sake of completeness, we also include the explanation of MPPI for obtaining the Planning policy.

B.1 Planning policy.

In this paper, we follow the same iterative planning process explained in Section 3 for **MPPI-based Reinforcement Learning**, where the Planning policy is iteratively refined with the help of a learned sampling policy and its associated Bootstrap action-value function. We maintain the same world model loss for the environment state encoder $h_{\theta_h}(\boldsymbol{s})$, dynamics model $p_{\theta_d}(\boldsymbol{z})$, and reward function model $r_{\theta_r}(\boldsymbol{z}, \boldsymbol{a})$ over latent representations from Hansen et al. (2024).

At each time step t, we start planning by encoding the current state of the environment $\boldsymbol{z}_t = h_\theta(\boldsymbol{s}_t)$. Then we sample simulated trajectories of horizon H, sampling $n_{\pi_{s_\theta}}$ times actions from the learned sampling policy π_{s_θ} and $M - n_{\pi_\theta}$ times from the Planning policy. The Planning policy is a Gaussian open-loop control sequence with mean: $\bar{a}_{0:H-1} = (\bar{a}_0, \ldots, \bar{a}_{H-1})$, and every sample being computed by $a_t^{(i)} = \bar{a}_t + \epsilon_t^{(i)}, \quad \epsilon_t^{(i)} \sim \mathcal{N}(0, \sigma_t I)$. The sequence is always initialized with variance σ^2_{max}, and the mean $\bar{a}_t$ with the 1-step shifted mean except for the start of the episode where zero-mean is used. M noisy trajectories are simulated $z_{t+1}^{(i)} \sim p\big(z_{t+1} \mid z_t^{(i)}, a_t^{(i)}\big)$ and evaluated according to its H-step estimated return:

$$\hat{Q}(z_0, a_{0:H}^{(i)}) = \sum_{t=0}^{H-1} \gamma^t r_{\theta_r}(z_t, a_t^{(i)}) + \gamma^H Q_{\theta_Q}^{\pi_{\theta_s}}(z_H, a_H^{(i)}) \tag{10}$$

After selecting the K-top performing samples, the MPPI update follows from a path-integral (desirability) transform:

$$\bar{a}_t \leftarrow \bar{a}_t + \sum_{i=1}^{K} w_i\, \epsilon_t^{(i)}, \quad \sigma_t = \sqrt{\frac{\sum_{i=1}^{K} w_i \left(\epsilon_t^{(i)}\right)^2}{\sum_{i=1}^{K} w^i}} \tag{11}$$

$$w_i = \frac{\exp\big(-\frac{1}{\beta}(Q(z_0, a_{0:H}^{(i)}) - \max_{i'} Q(z_0, a_{0:H}^{(i')}))\big)}{\sum_{j=1}^{K} \exp\big(-\frac{1}{\beta}(Q(z_0, a_{0:H}^{(j)}) - \max_{i'} Q(z_0, a_{0:H}^{(i')}))\big)},$$

where $\beta > 0$ is the temperature parameter, and controls how much the importance sampling scheme weights the optimal cost trajectory versus the others. After a fixed number of iterations, the planning procedure is terminated and a trajectory is sampled from the final return-normalized distribution over action sequences. Planning is done at each decision step, and only the first action of the sampled trajectory, a_0, is executed to produce a feedback policy. We denote the resulting Planning policy over the first step, obtained after a fixed number of MPPI iterations, by: $\pi_{\mathrm{P}} = \mathcal{N}(\bar{a}_0, \sigma_0 I)$, with p being the transition model, and $\bar{a}_{init}$ the initialization mean control sequence. After interacting with the environment, the transition information and Planning policy are added to a replay buffer, i.e. $(s, a_0, s', r, \bar{a}_0, \sigma_0) \longrightarrow \mathcal{D}$.

B.2 Adaptive prior policy updates.

To improve the sampling policy using KL-regularized RL, we need a policy prior π_p representing the current Planning policy to act as a reference. To represent the current Planning policy we can straightforwardly use the Planning policy samples stored in the replay buffer or, as shown in Section 4, an intermediate policy π_{θ_p}. We train this intermediary policy by either minimizing the reverse KL divergence:

$$J(\theta_p) = \sum_{t'=t}^{H-1} \frac{\rho^{t'-t}}{H} \frac{\mathrm{KL}[\pi_{\theta_p}(\cdot \mid z_{t'}) \parallel \pi_P(\cdot \mid z_{t'})]}{\max(1, S_p)}, \tag{12}$$

or, as an example of a straightforward alternative, the forward KL divergence:

$$J(\theta_p) = \sum_{t'=t}^{H-1} \frac{\rho^{t'-t}}{H} \frac{\mathrm{KL}[\pi_P(\cdot \mid z_{t'}) \parallel \pi_{\theta_p}(\cdot \mid z_{t'})]}{\max(1, S_p)}, \tag{13}$$

where S_p is an adaptive scale parameter that tracks the difference between the 5^{th} and 95^{th} percentiles of the KL divergence. This is often use

B.3 Action value function and policy updates.

Planning policy improvement relies on improving the sampling policy, π_{θ_s}, and updating its associated bootstrap action value function, $Q_{\theta_Q}^{\pi_{\theta_s}}$. Every n_d time steps, a batch of n_b trajectories of horizon H is drawn from the replay buffer $\mathcal{D}$. The action value function $Q_{\theta_Q}^{\pi_{\theta_s}}$ is updated by minimizing its TD-error at each time step over the horizon H, with a decaying parameter ρ to account for prediction error over the latent space predictions. In the following, we denote by $\pi_{\theta_s}(z)$ the learned sampling policy probability distribution over actions u conditional on the latent representation $z = h_{\theta_h}(s)$, leaving $\pi_{\theta_s}(u|z)$ to denote the probability of sampling u under the learned sampling policy.

$$J(\theta_Q) = \sum_{t'=t}^{H-1} \frac{\rho^{t'-t}}{H} \mathrm{CE}(Q_{\theta_Q}^{\pi_{\theta_s}}(z_{t'}, a_{t'}), \hat{Q}^{\pi_{\theta_s}}(z_{t'}, a_{t'})) \tag{14}$$

$$\hat{Q}^{\pi_{\theta_s}}(z_{t'}, a_{t'}) = r_t + \gamma Q_{\theta_Q^-}^{\pi_{\theta_s}}(z_{t'+1}, \tilde{a})|_{\tilde{a}\sim\pi_{\theta_s}(a|z_{t'+1})} \tag{15}$$

Where θ_Q and θ_Q^- are the parameters of the action value function and the target action value function. As explained in Hansen et al. (2024), the TD-error is tracked by the cross-entropy error between action-value logit representations and the two-hot vector encoding of the target. Under the assumption that the action-value function is correctly approximated, the planning policy is a maximum a posteriori estimate over the learned sampling distribution π_{θ_s}. Therefore, the planning policy can be intuitively interpreted as a policy improvement step over the current learned policy Sutton & Barto (2018).

The learned sampling policy update is designed to move the policy towards maximizing the expected return while ensuring its associated trajectory distribution remains close to the prior trajectory distribution, which is induced by the planning policy. This leads to the following KL-regularized action value function loss:

$$J(\tilde{\theta}_Q) = \sum_{t'=t}^{H-1} \frac{\rho^{t'-t}}{H} \mathrm{CE}(Q_{\tilde{\theta}_Q}^{\pi_{\theta_s},\lambda}(z_{t'}, a_{t'}), \hat{Q}^{\pi_{\theta_s},\lambda}(z_{t'}, a_{t'})) \tag{16}$$

$$\hat{Q}^{\pi_{\theta_s},\lambda}(z_{t'}, a_{t'}) = r_t + \gamma \left(Q_{\tilde{\theta}_Q^-}^{\pi_{\theta_s},\lambda}(z_{t'+1}, \tilde{a})|_{\tilde{a}\sim\pi_{\theta_s}(z_{t'+1})} - \lambda \frac{\mathrm{KL}[\pi_{\theta_s}(\cdot|z_{t'+1}) \parallel \pi_{\theta_p}(\cdot|z_{t'+1})]}{\max(1, S_{KL})} \right) \tag{17}$$

and the following policy loss:

$$J(\theta_s) = \sum_{t'=t}^{H-1} \frac{\rho^{t'-t}}{H} \left(\lambda \frac{\mathrm{KL}[\pi_{\theta_s}(z_{t'}) \parallel \pi_{\theta_p}(z_{t'})]}{\max(1, S_{KL})} - \frac{Q_{\tilde{\theta}_Q}^{\pi_{\theta_s},\lambda}(z_{t'}, \tilde{a})|_{\tilde{a}\sim\pi_{\phi_\pi}(z_{t'})}}{\max(1, S_Q)} - \alpha\mathcal{H}(\pi_{\theta_s}(z_t)) \right), \tag{18}$$

where S_i, $i \in \{KL, Q\}$, is an adaptive scale parameter that tracks the difference between the 5^{th} and 95^{th} percentiles of each loss term. Since the values of both terms differ by multiple degrees of magnitude, scaling them enables more robust control, through the hyperparameter λ, over the trade-off between expected return maximization and mimicking the policy prior distribution.

It is important to note that, due to its potential to reach very high values, which may negatively affect action value learning and, consequently, exploration, the KL term, both in action value target and sampling policy update, is often scaled by S_{KL} in practice.

B.4 Co-dependence between the learned policy and the planning policy.

During the first steps of training, the replay buffer needs to be filled, and the planning policy suffers from low quality since both $Q^{\pi_{\theta_s}}_{\theta_Q}, \pi_{\theta_s}$ are untrained. This is why it is important to make sure the bootstrap action value function is properly trained before updating all the other components. Therefore, we follow a pretraining phase during the first N_s steps, where only the untrained sampling policy π_{θ_s} interacts with the environment with no parameter updates. Then, before proceeding to update all parameters as explained in Section 4, we update all model parameters and the bootstrapping action value function ($Q^{\pi_{\theta_s}}_{\theta_Q}$) N_s times. To prevent unnecessary exploration bias, the planning policy samples stored during this phase are zero-mean diagonal Gaussians with maximum standard deviation σ_{max}. This ties with another relevant implementation detail. Due to the planning policy depending on an ever-evolving policy distribution, planning policy samples saved in the replay buffer eventually become outdated. To alleviate this problem, we employ *lazy reanalyze* (Wang et al., 2025), which takes inspiration from the Wang et al. (2024b); Schrittwieser et al. (2021) to periodically update partially a subset of the planning distributions sampled from the replay buffer.

B.5 Architecture and Framework

In this work, we build upon the partial implementation of TD-MPC2 in JAX (Bradbury et al., 2018) by Flandermeyer (2024a). We inherit all architectural choices from TD-MPC2. The architecture of $Q^{\pi_{\theta_s},\lambda}_{\hat{\theta}_Q}$ follows the same design of its counterpart $Q^{\pi_{\theta_s}}_{\theta_Q}$. Despite updating an additional policy and action value function, training times do not differ significantly from the baselines.

B.6 Baselines.

For our experiments, we employ the implementations in JAX (Flandermeyer, 2024a;b), developed with the collaboration of the original authors, since they reproduce the results from the original paper while increasing the computation speed.

C PO-MPC ALGORITHM

Algorithm 2 PO-MPC

```
Require: Replay buffer $\mathcal{B}$, Data-to-update ratio $n_{d2u}$, and Reanalyze interval $k$.
    Initialize: $\pi_{\theta_s}$, $Q^{\pi}_{\theta_Q}$, $Q^{\pi_{\theta_s},\lambda}_{\tilde{\theta}_Q}$.
    Initialize MDP model: $\tilde{\mathcal{M}} := (h_{\theta_h}, p_{\theta_d}, r_{\theta_r})$.
    Initialize planning priors: $a_{init}$, $\sigma_{max}$
 1: n_updates $= 0$
 2: for t=1,2,...,T do
 3:     // Environment interaction
 4:     $z_t \leftarrow h_{\theta_h}(s_t)$
 5:     // Planning Policy (Section B.1)
 6:     $a_t, \bar{a}_{t:t+H}, \sigma_{t:t+H} \leftarrow$ MPPI$_{\tilde{\mathcal{M}}}(z_t|\pi_{\theta_s}, Q^{\pi_{\theta_s}}_{\theta_Q}, \bar{a}_{init})$
 7:     $s_{t+1}, r_t \leftarrow$ environment_step$(s_t, a_t)$
 8:     $\mathcal{B} \cup \{s_t, a_t, r_t, s_{t+1}, \bar{a}_t, \sigma_t\}$
 9:     // Gradient updates.
10:     if t (mod $n_{d2u}$) $== 0$ then
11:         n_updates $\leftarrow$ n_updates $+ 1$
12:         $\mathcal{D}_{n_b} := \{s_{t'}, a_{t'}, r_{t'}, s_{t'+1}, \bar{a}_{t'}, \sigma_{t'}\}^{1:n_b}_{t':t'+H} \sim \mathcal{D}$
13:         $z_{t':t'+H} \leftarrow h_{\theta_h}(s_{t':t'+H})$
14:         // Update Planning samples via Lazy reanalyze as in Wang et al. (2025).
15:         if n_updates (mod $k$) $== 0$ then
16:             $\mathcal{D}_{n_b^r} \sim \mathcal{D}_{n_b}$, $n_b^r \leq n_b$
17:             $a_{t'}, \bar{a}_{t':t'+H}, \sigma_{t':t'+H} \leftarrow$ MPPI$_{\tilde{\mathcal{M}}}(z_{t'}|\pi_{\theta_s}, Q^{\pi_{\theta_s}}_{\theta_Q}, \mathbf{0})$
18:             $\mathcal{D}_{n_b^r} \leftarrow \{s_{t'}, u_{t'}, r_{t'}, a_{t'+1}, \bar{a}_{t'}, \sigma_{t'}\}$
19:         end if
20:         $\pi_P(a_{t'}|z_{t'}) \leftarrow \mathcal{N}(\bar{a}_{t'}, \sigma^2_{t'} I)$
21:         Update MDP model: $h_{\theta_h}, d_{\theta_d}, r_{\theta_r}$ as in Hansen et al. (2024).
22:         Update Bootstrap action value function: $Q^{\pi_{\theta_s}}_{\theta_Q}$ (Equation 14)
23:         Update Policy prior: $\pi_{\theta_p}$ (Equation 8 or Equation 9
24:         Update KL regularized action value function: $Q^{\pi_{\theta_s},\lambda}_{\tilde{\theta}_Q}$ (Equation 16)
25:         Update Sampling Policy $\pi_{\theta_s}$ (Equation 18)
26:         $\theta^-_Q \leftarrow \tau\theta_Q + (1-\tau)\theta^-_Q$
27:         $\tilde{\theta}^-_Q \leftarrow \tau\tilde{\theta}_Q + (1-\tau)\tilde{\theta}^-_Q$
28:     end if
29: end for
```

D Additional Results

D.1 Results in DMControl Suite

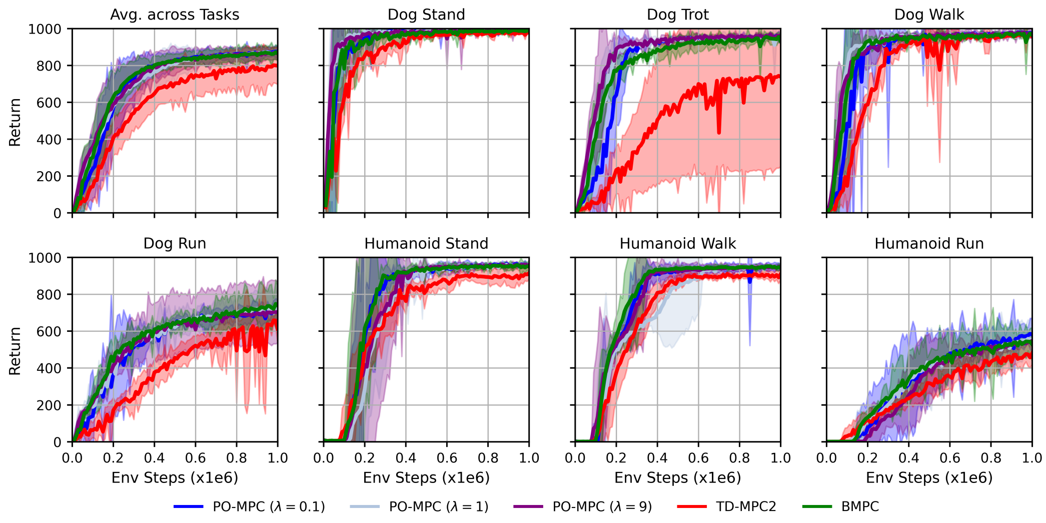

Figure 5: Performance comparison of PO-MPC and the baselines on 7 state-based high-dimensional control tasks from DMControl Suite (Tassa et al., 2018). Mean of 3 runs; shaded areas are 95% confidence intervals. In the top left, we visualize results averaged across all 7 tasks.

D.2 Intermediate Policy Prior Performance

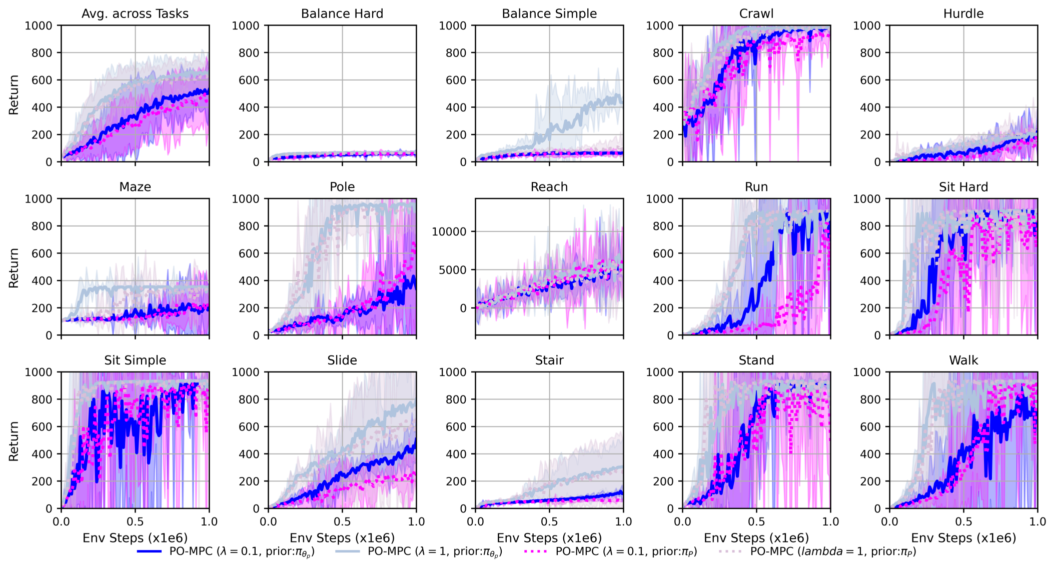

Figure 6: Performance comparison in 14 state-based high-dimensional control tasks from HumanoidBench (Sferrazza et al., 2024). Mean of 3 runs; shaded areas are 95% confidence intervals. In the top left, we visualize results averaged across all tasks except for *Reach*, which has a different return range. We observe that using the intermediate policy not only does not harm the performance but also enhances it in some tasks.

D.2.1 Shielding Effect of the Policy Prior

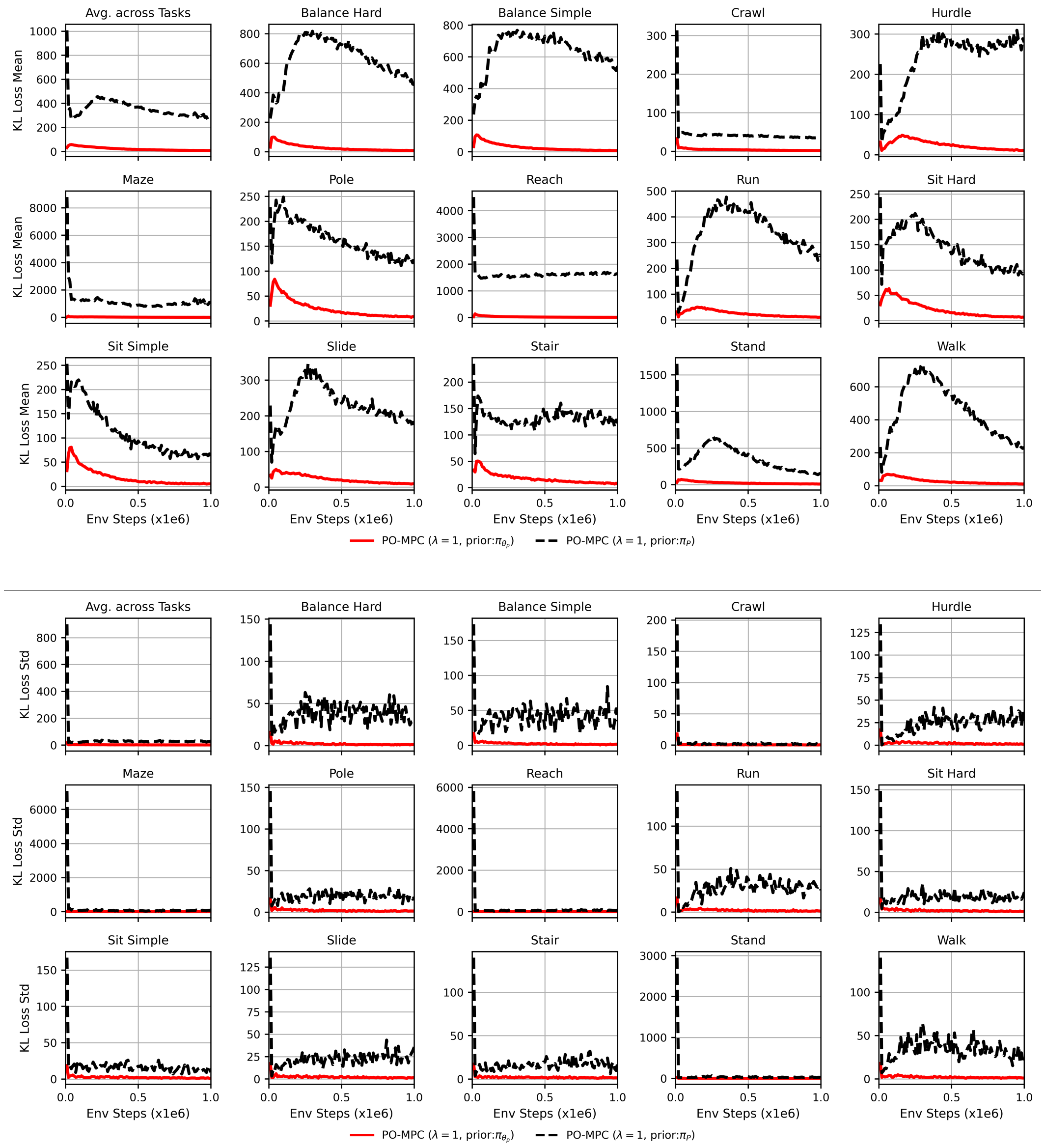

Figure 7: **Top:** Mean and, **Bottom:**Standard deviation of the KL divergence term in Equation 18 for both PO-MPC using an intermediate policy prior and the Planning policy. Experiments are done in the HumanoidBench Locomotion suite (Sferrazza et al., 2024). Mean of 3 runs. We show empirical evidence on how the mean and standard deviation of the KL term are significantly larger when the Planning policy samples are used instead of the intermediate policy prior. This shows that the intermediate policy prior effectively shields the sampling policy updates from high variance being introduced by outdated Planning policy samples stored in the replay buffer. Similar results are obtained across different values of λ, and we present results for $\lambda = 1$ for the sake of clarity.

D.2.2 Training Policy Prior with Reverse KL vs Forward KL Loss

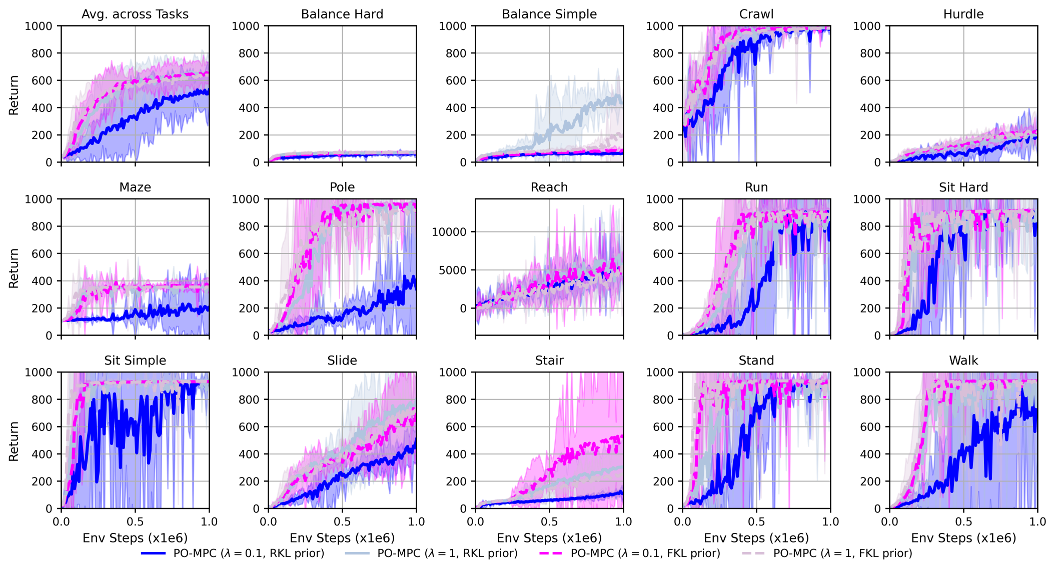

Figure 8: Performance comparison in 14 state-based high-dimensional control tasks from HumanoidBench Locomotion suite (Sferrazza et al., 2024). Mean of 3 runs; shaded areas are 95% confidence intervals. In the top left, we visualize results averaged across all tasks except for *Reach*, which has a different return range. We observe that training the policy prior with the Forward KL divergence instead of the Reverse KL divergence can help in finding a solution faster in some tasks but may be detrimental in others requiring more precision such as *Balance Simple*.

E Proofs

We present proof of convergence of both the KL-regularized Policy Evaluation step and policy improvement. We follow closely the same proofs for Max. Entropy RL from Haarnoja et al. (2018) since it is a particular case of KL-regularized RL. Substituting π_p by the uniform distribution over the action space $\mathcal{A}$ recovers the proof from Haarnoja et al. (2018). Note that the only additional requirement needed for the KL-regularized version is $\frac{\pi(a_t|s_t)}{\pi_p(a_t|s_t)}$ being determined almost everywhere.

Lemma E.1. *(KL-regularized Policy Evaluation). Given a policy and policy prior $\pi, \pi_p \in \Pi$. Let the KL-regularized Bellman backup operator:*

$$\mathcal{T}^{\pi} Q^{\pi,\lambda}(s_t, a_t) \coloneqq r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}[V^{\pi,\lambda}(s_{t+1})] \tag{19}$$

where

$$V^{\pi,\lambda}(s_t) = \mathbb{E}_{a_t\sim\pi(\cdot|s_t)}[Q^{\pi,\lambda}(s_t,a_t) - \lambda[\log \pi(a_t|s_t) - \log \pi_p(a_t|s_t)]], \tag{20}$$

and a mapping $Q_0 : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$, where $\mathcal{A}$ is bounded and $\frac{\pi(a_t|s_t)}{\pi_p(a_t|s_t)}$ is determined almost everywhere, we define $Q^{\pi,\lambda}_{k+1} = \mathcal{T}^{\pi} Q^{\pi,\lambda}_{k}$. Then the sequence $Q^{\pi,\lambda}_{k}$ will converge to the KL regularized action-value of π as $k \to \infty$.

Proof. Define the KL augmented reward as:

$$r_\pi \coloneqq r(s_t, a_t) - \gamma\lambda \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}[\mathrm{KL}[\pi(\cdot \mid s_{t+1}) \parallel \pi_p(\cdot \mid s_{t+1})]] \tag{21}$$

and rewrite the update as:

$$Q^{\pi,\lambda}(s_t, a_t) \leftarrow r_\pi(s_t, a_t) + \gamma \mathbb{E}_{\substack{s_{t+1}\sim p(\cdot|s_t,a_t),\\ a\sim\pi(\cdot|s_{t+1})}}[Q^{\pi,\lambda}(s_{t+1}, a_{t+1})] \tag{22}$$

Then we can apply the standard convergence results for policy evaluation from Sutton & Barto (2018). A bounded action space $\mathcal{A}$ and KL-divergence between π and π_p are necessary assumptions to guarantee that the augmented reward r_π is bounded. □

Lemma E.2. *(KL-regularized Policy Improvement) Let $\pi_{old} \in \Pi$ and let π_{new} be the optimizer of the minimization problem defined as:*

$$\pi_{new} = \arg\min_{\pi'\in\Pi} KL[\pi'(\cdot|s_t) \parallel \frac{\exp(\frac{1}{\lambda} Q^{\pi_{old},\lambda}(s_t,\cdot))}{Z^{\pi_{old}}(s_t)} \pi_p(\cdot|s_t)] = \arg\min_{\pi'\in\Pi} \mathcal{J}_{\pi_{old}}(\pi'(\cdot|s_t)) \tag{23}$$

Then $Q^{\pi_{new},\lambda}(s_t,a_t) \geq Q^{\pi_{old},\lambda}(s_t,a_t), \forall (s_t,a_t) \in \mathcal{S}\times\mathcal{A}$ with $|\mathcal{A}| < \infty$ being bounded.

Proof. Let $\pi^{old} \in \Pi$, and $Q^{\pi_{old},\lambda}$, $V^{\pi_{old},\lambda}$, its respective KL-regularized action-value and value function. Then we define:

$$\begin{aligned}\pi_{new} &= \arg\min_{\pi'\in\Pi} \mathcal{J}_{\pi_{old}}(\pi'(\cdot|s_t)) \\ &= \arg\min_{\pi'\in\Pi} \mathrm{KL}[\pi'(\cdot|s_t) \parallel \exp(\lambda^{-1} Q^{\pi_{old},\lambda}(s_t,\cdot) - \log Z^{\pi_{old}}(s_t) + \log \pi_p(\cdot|s_t))]\end{aligned} \tag{24}$$

It must be the case that $\mathcal{J}_{\pi_{old}}(\pi_{new}(\cdot|s_t)) \leq \mathcal{J}_{\pi_{old}}(\pi_{old}(\cdot|s_t))$, since we can always choose $\pi_{new} = \pi_{old} \in \Pi$. Hence,

$$\begin{aligned}&\mathbb{E}_{a_t\sim\pi^{new}}\left[\log \pi^{new}(a_t|s_t) - \lambda^{-1} Q^{\pi_{old},\lambda}(s_t,a_t) + \log Z^{\pi_{old}}(s_t) - \log \pi_p(a_t|s_t)\right] \leq \\ &\mathbb{E}_{a_t\sim\pi^{old}}\left[\log \pi^{old}(a_t|s_t) - \lambda^{-1} Q^{\pi_{old},\lambda}(s_t,a_t) + \log Z^{\pi_{old}}(s_t) - \log \pi_p(a_t|s_t)\right].\end{aligned} \tag{25}$$

Since $Z^{\pi^{old}}$ does not depend on a_t, equation 25 reduces to:

$$\begin{aligned}&\mathbb{E}_{a_t\sim\pi^{new}}\left[Q^{\pi_{old},\lambda}(s_t,a_t) - \lambda \log \frac{\pi^{new}(a_t|s_t)}{\pi_p(a_t|s_t)}\right] \geq \\ &\mathbb{E}_{a_t\sim\pi^{old}}\left[Q^{\pi_{old},\lambda}(s_t,a_t) - \lambda \log \frac{\pi^{old}(a_t|s_t)}{\pi_p(a_t|s_t)}\right] = V^{\pi_{old},\lambda}(s_t).\end{aligned} \tag{26}$$

Then, unrolling $Q^{\pi_{old},\lambda}(s_t, a_t)$ and applying the bound in equation 26 results in:

$$\begin{aligned}
Q^{\pi_{old},\lambda}(s_t, a_t) &= r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}[V^{\pi_{old},\lambda}(s_{t+1})] \\
&\leq r(s_t, a_t) + \gamma \mathbb{E}_{\substack{s_{t+1}\sim p(\cdot|s_t,a_t) \\ a_{t+1}\sim\pi_{new}(\cdot|s_{t+1})}} \left[Q^{\pi_{old},\lambda}(s_{t+1}, a_{t+1}) - \lambda \log \frac{\pi^{new}(a_{t+1}|s_{t+1})}{\pi_p(a_{t+1}|s_{t+1})} \right] \\
&= r(s_t, a_t) + \gamma \mathbb{E}_{\substack{s_{t+1}\sim p(\cdot|s_t,a_t) \\ a_{t+1}\sim\pi_{new}(\cdot|s_{t+1})}} \left[r(s_{t+1}, a_{t+1}) - \lambda \log \frac{\pi^{new}(a_{t+1}|s_{t+1})}{\pi_p(a_{t+1}|s_{t+1})} \right. \\
&\qquad\qquad \left. + \gamma \mathbb{E}_{s_{t+2}\sim p(\cdot|s_{t+1},a_{t+1})}[V^{\pi_{old},\lambda}(s_{t+2})] \right] \\
&\;\vdots \\
&\leq Q^{\pi_{new},\lambda}(s_t, a_t) \qquad (27)
\end{aligned}$$

Convergence to $Q^{\pi_{new},\lambda}$ follows from Lemma E.1

□

Theorem E.3. *(KL-regularized policy iteration). Repeated application of KL-regularized policy evaluation and KL-regularized policy improvement to any $\pi \in \Pi$ converges to a policy π^* such that $Q^{\pi^*,\lambda}(s_t, a_t) \geq Q^{\pi,\lambda}(s_t, a_t)$ for all $\pi \in \Pi$ and $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$, assuming $|\mathcal{A}| < \infty$.*

Proof. The proof follows the same reasoning from Theorem 1 in Haarnoja et al. (2018). Let policy π_i be the policy at iteration i. By Lemma E.2, the sequence $Q^{\pi_i,\lambda}$ is monotonically increasing. Since $Q^{\pi,\lambda}$ is bounded above for $\pi \in \Pi$, since both reward and KL-divergence are bounded, the sequence converges to some π^*. To show that π^* is optimal, it must be the case that, at convergence, $J_{\pi^*}(\pi^*(\cdot|s_t)) < J_{\pi^*}(\pi(\cdot|s_t)), \forall \pi \in \Pi, \pi \neq \pi^*$. Using the same iterative argument as in the proof of Lemma E.2, we get $Q^{\pi^*,\lambda}(s_t, a_t) > Q^{\pi,\lambda}(s_t, a_t), \forall (s_t, a_t) \in \mathcal{S} \times \mathcal{A}$, meaning the KL-regularized value of any other policy in Π is lower than that of the converged policy. Hence π^* is optimal in Π. □

F Background and Connection to Recent Work

There is some recent work that, while also within MPPI-based RL, cannot be unified within the proposed framework as seamlessly since they constitute an approximation of our theoretical framework(Lin et al., 2025), forgoing the monotonic improvement guarantees shown in Appendix E, or optimize a different cost function (Zhuang et al., 2025). We will start this section by briefly reviewing the background of RL cast as a probabilistic inference problem, which ultimately boils down to the KL-regularized RL formulation, since then the main differences with respect to these works will become clearer.

F.1 RL cast as probabilistic inference

We follow the same reasoning as in Levine (2018). Let $\tau := (s_0, a_0, s_1, a_1, ..., s_T)$ a trajectory across the joint state-action space $\mathcal{S} \times \mathcal{A}$, the probability of a trajectory τ given a parametric policy π_θ:

$$p(\tau; \pi_\theta) = \rho_0(s_0) \prod_{t=0}^{T} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t) \tag{28}$$

Next, we assume that the probability of (s, a) being part of an optimal trajectory is proportional to $\exp(\frac{1}{\lambda} r(s_t, a_t))$. Then, it follows that the joint probability of a trajectory given a prior policy π_p and that trajectory being optimal is given by:

$$p(\tau, O_{0:T}; \pi_p) = \rho_0(s_0) \frac{1}{Z_t} \exp(\frac{1}{\lambda} R_t) \prod_{t=0}^{T} p(s_{t+1}|s_t, a_t)\pi_p(a_t|s_t) \tag{29}$$

Where $\rho_0(s_0)$ is the probability of starting at state s_0, $p(s_{t+1}|s_t, a_t)$ is the transition probability, and λ is the inverse temperature. Note that λ trades off the effect of the policy prior and the trajectory's return in the joint probability distribution. The event of τ being an optimal trajectory is represented by $O_{0:T}$ (Levine, 2018), and $R_t = \sum_{t=0}^{T} r(s_t, a_t)$. Then, the KL-regularized RL formulation presented in Equation 5 stems from minimizing **the reverse KL-divergence between** $p(\tau; \pi_\theta)$ **and** $p(\tau, O_{0:T}; \pi_p)$:

$$\begin{aligned}
\theta^* &= \arg\min_\theta \mathrm{KL}[p(\tau; \pi_\theta) \parallel p(\tau, O_{0:T}; \pi_p)] && (30)\\
&= \arg\min_\theta \mathbb{E}_{\substack{s_0 \sim \rho_0, a_t \sim \pi_\theta(\cdot|s_t),\\ s_{t+1} \sim p(\cdot|s_t, a_t)}} \Big[\log \frac{p(\tau; \pi_\theta)}{p(\tau, O_{0:T}; \pi_p)} \Big] && (31)\\
&= \arg\max_\theta \mathbb{E}_{\substack{s_0 \sim \rho_0, a_t \sim \pi_\theta(\cdot|s_t),\\ s_{t+1} \sim p(\cdot|s_t, a_t)}} \Big[\sum_{t=0}^{T-1} \big[r(s_t, a_t) - \lambda \log \frac{\pi_\theta(a_t \mid s_t)}{\pi_p(a_t \mid s_t)} \big] \Big] && (32)\\
&= \arg\max_\theta \mathbb{E}_{\substack{s_0 \sim \rho_0, a_t \sim \pi_\theta(\cdot|s_t),\\ s_{t+1} \sim p(\cdot|s_t, a_t)}} \Big[\sum_{t=0}^{T-1} \big[r(s_t, a_t) - \lambda \mathrm{KL}[\pi_\theta(\cdot \mid s_t) \parallel \pi_p(\cdot \mid s_t)] \big] \Big], && (33)
\end{aligned}$$

Which results in the step-wise objective described in Equation 6 with the KL-regularized action-value function $Q^{\pi_\theta, \lambda}$ defined in Equation 7.

F.2 TD-M(PC)^2 (Lin et al., 2025)

The previous policy update is core to the PO-MPC framework and is guaranteed to monotonically improve under the assumptions given in Appendix E. It also includes existing methods in the literature (i.e. TD-MPC2 (Hansen et al., 2024), BMPC (Wang et al., 2025)) when learning the sampling policy π_{θ_s} under different values of the hyperparameter λ and a policy prior that represents the planner policy, either from data stored in the replay buffer $\pi_p = \pi_P$ or a proxy distribution $\pi_p = \pi_{\theta_p}$.

However, it adds further complexity since it requires learning an additional action value function $Q^{\pi_\theta,\lambda}$. Therefore, **recent methods such as TD-M(PC)^2**, choose to forgo theoretical guarantees in favor of simplicity by using the unregularized Q^π, which is still bound to perform well as long as the policy remains close to the planner. This is enforced by maximizing:

$$\mathbb{E}_{a\sim\pi_{\theta_s}}[Q^{\pi_{\theta_s}}(z_t,a_t)-\alpha\log\pi(\cdot\mid z_t)+\beta\log\pi_p(\cdot\mid z_t)] \tag{34}$$

Where α is the hyperparameter regulating entropy maximization and β modulates the cross-entropy These hyperparameters are often tuned so that $\alpha \ll \beta$, which allows to further develop this expression into:

$$\mathbb{E}_{a_t\sim\pi_{\theta_s}}\Big[Q^{\pi_{\theta_s}}(z_t,a_t)-\alpha\log\pi_{\theta_s}(a_t\mid z_t)+\beta\log\pi_p(a_t\mid z_t)] = \mathbb{E}_{a_t\sim\pi_{\theta_s}}[Q^{\pi_{\theta_s}}(z_t,a_t)+(\beta-\alpha)\log\pi_{\theta_s}(a_t\mid z_t)-\beta\log\frac{\pi_{\theta_s}(a_t\mid}{\pi_p(a_t\mid} \tag{35}$$

Which can be seen as learning a policy that maximizes the action value function $Q^{\pi_{\theta_s}}$ while minimizing both KL-divergence with respect to π_p, and entropy, since $\beta-\alpha$ is positive under $\alpha\ll\beta$.

F.3 TD-MPBC (Zhuang et al., 2025)

Other recent works in MPPI-based RL cannot be included within our formulation because the starting loss function being minimized is different. The policy update from TD-MPBC (Zhuang et al., 2025) comes from minimizing **the forward KL-divergence between** $p(\tau;\pi_\theta)$ **and** $p(\tau,O_{0:T};\pi_p)$:

$$\begin{aligned}
\theta^* &= \arg\min_\theta \mathrm{KL}[p(\tau,O_{0:T};\pi_p)\parallel p(\tau;\pi_\theta)] && (36)\\
&= \arg\min_\theta \mathbb{E}_{(s_t,a_t)\sim p(\tau,O_{0:T};\pi_p)}\Big[\log\frac{p(\tau,O_{0:T};\pi_p)}{p(\tau;\pi_\theta)}\Big] && (37)\\
&= \arg\max_\theta \mathbb{E}_{(s_t,a_t)\sim p(\tau,O_{0:T};\pi_p)}\Big[\log p(\tau;\pi_\theta)\Big] && (38)\\
&= \arg\max_\theta \mathbb{E}_{(s_t,a_t)\sim p(\tau,O_{0:T};\pi_p)}\Big[\sum_{t=0}^{T-1}\log\pi_\theta(a_t\mid s_t)\Big] && (39)\\
&= \arg\max_\theta \mathbb{E}_{(s_t,a_t)\sim p(\tau;\pi_p)}\Big[\sum_{t=0}^{T-1}\frac{1}{Z_t}\exp(\frac{1}{\lambda}R_t)\log\pi_\theta(a_t\mid s_t)\Big] && (40)\\
&= \arg\max_\theta \mathbb{E}_{(s_t,a_t)\sim p(\tau;\pi_p)}\Big[\sum_{t=0}^{T-1}\exp(\frac{1}{\lambda}R_t-\log Z_t)\log\pi_\theta(a_t\mid s_t)\Big] && (41)\\
& && (42)
\end{aligned}$$

Since in this case π_p can be an arbitrary policy, we can estimate $\mathbb{E}_{(s_t,a_t)\sim p(\tau;\pi_p)}$ through Monte Carlo estimation. Choosing $\lambda=G$ and $\log Z_t=1$, we recover the Behavior cloning regularization term from TD-MPBC:

$$\arg\max_\theta \sum_{(s_t,a_t,R_t)\sim\mathcal{D}}\exp\Big(\frac{R_t-G}{G}\Big)\log\pi_\theta(a_t\mid s_t) \tag{43}$$

This term has high variance, since it depends on the returns R_t from the full trajectory. It is also potentially sparse, as it depends on the version of the planner available when sampling the trajectory and selecting high-return trajectory samples from the replay buffer. This is possibly why the main learning signal in TD-MPBC originates from using the policy updates from TD-MPC2 (equivalent to PO-MPC with $\lambda=0$), with the term in Equation 43 serving as a regularization term.

G Theoretical Justification over the Adaptive Prior

This section yields a theoretical justification for why using a learned policy prior that minimizes the KL divergence with the previously stored planner policy samples from the replay buffer, instead of using these directly, may decrease gradient variance from the sampling policy updates. First, we introduce the formal definition for the gradient related to the regularization term that results from using each representation of the planner policy. Then we compute the variance of each and compare them.

Let us sample transitions from 1 to K, K being the current time step, and let $\mathcal{K} := \{1, 2, ..., K\}$ the set of previous time steps. Let $(s_k, a_k, s_{k+1}, r_k, \mu_k, \sigma_k)$ be the transition stored at time step k, where μ_k, σ_k are the mean and standard deviation of the planner policy computed at time step k: $\pi^k := \mathcal{N}(\mu_k, \sigma_k^2 I)$.

Assume that we can isolate $N << K$ samples that share the same state s and compute the mean KL-divergence between the parametric sampling policy we want to update, π_θ (where $\theta = \theta_s$ for simplicity), and some generic prior policy, π_p, that represents the planner. Then the regularization term in the loss function is:

$$J(\theta) = \frac{1}{N} \sum_{i=0}^{N} \mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[\log \pi_\theta(a|s) - \log \pi_p(a|s)], \tag{44}$$

with its gradient being:

$$\begin{aligned} \nabla J(\theta) &= \nabla \frac{1}{N} \sum_{i=0}^{N} \mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[\log \pi_\theta(a|s) - \log \pi_p(a|s)] \\ &= \frac{1}{N} \sum_{i=0}^{N} \nabla \mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[\log \pi_\theta(a|s)] - \frac{1}{N} \sum_{i=0}^{N} \nabla \mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[\log \pi_p(a|s)] \end{aligned} \tag{45}$$

We isolate and focus on the cross-entropy term since it is the only one that depends on π_p:

$$g(\theta) = \frac{1}{N} \sum_{i=0}^{N} \nabla \mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[\log \pi_p(a|s)] = \frac{1}{N} \sum_{i=0}^{N} \mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[\nabla \log \pi_\theta(a|s) \log \pi_p(a|s)] \tag{46}$$

G.1 Expected value

Previously stored planner policies. Let π^{k_i} be policy planner stored at time step $k_i \in \mathcal{K}$ where $i = 1, ..., N$, and let k_i be sampled uniformly from $\mathcal{K}$. Substituting π_p in Equation 46 yields:

$$g_1(\theta) = \mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[\nabla_\theta \log \pi_\theta(a|s)(\frac{1}{N} \sum_{i=1}^{N} \log \pi^{k_i}(a|s))] \tag{47}$$

Next, the expected value of the gradient is:

$$\mathbb{E}_k[g_1(\theta)] = \mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[\nabla_\theta \log \pi_\theta(a|s) \mathbb{E}_k[\log \pi^k(a|s)]] \tag{48}$$

Learned policy prior. Now let us develop the effect of learning first a prior from these samples, using it as an alternative to regularize the sampling policy π_θ. Let π_{θ_p} be learned policy prior that iteratively minimizes the reverse KL divergence between itself and the sequence of previously stored planner policies π^k:

$$\pi_{\theta_p} := \arg\min_{\pi} \frac{1}{K} \sum_{k=0}^{K} \mathbb{E}_{a \sim \pi_{\theta_p}}[\log \pi_{\theta_p}(a|s) - \log \pi^k(a|s)] = \mathbb{E}_k[\log \pi^k(a|s)]. \tag{49}$$

Then it follows that $\log \pi_{\theta_p} = \mathbb{E}_k[\log \pi^k(a|s)]$. This is the first important result since substituting π_p in Equation 46 now yields,

$$g_2(\theta) = \mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[\nabla_\theta \log \pi_\theta(a|s) \log \pi_{\theta_p}(a|s)] = \mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[\nabla_\theta \log \pi_\theta(a|s) \mathbb{E}_k[\log \pi^k(a|s)]], \tag{50}$$

Therefore, when using the learned policy prior, the mean of the gradient term that depends on π_p is theoretically equivalent to using previously stored samples: $\mathbb{E}_k[g_1] = \mathbb{E}_k[g_2]$.

G.2 Variance

Previously stored planner policies. The source of randomness for g_1 comes both from the sampled actions and the sampled planner policies. Thus, using the previous result, we can make the following decomposition:

$$\frac{1}{N}\sum_{i=1}^{N} \log \pi^{k_i}(a|s) = \mathbb{E}_k[\log \pi^k(a|s)] + \epsilon_N(a), \tag{51}$$

where $\mathbb{E}_k[\epsilon_N(a)] = 0$ and $\mathrm{Var}_k(\epsilon_N(a)) = \frac{1}{N}\mathrm{Var}_k(\log \pi^k(a|s))$

Then $g_1(\theta) = \mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[\nabla_\theta \log \pi_\theta(a|s)\mathbb{E}_k[\log \pi^k(a|s)]] + \mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[\nabla_\theta \log \pi_\theta(a|s)\epsilon_N(a)]$. Using the law of total variance and that $\epsilon_N(a)$ has zero mean we obtain:

$$\begin{aligned} \mathrm{Var}_{\substack{k \\ a\sim\pi_\theta}}(g_1(a)) &= \mathrm{Var}_{a\sim\pi_\theta}(\mathbb{E}_k[g_1|a]) + \mathbb{E}_{a\sim\pi_\theta}[\mathrm{Var}_k(g_1|a)] \\ &= \mathrm{Var}_{a\sim\pi_\theta}(\nabla_\theta \log \pi_\theta(a|s)\mathbb{E}_k[\log \pi^k(a|s)]) \\ &\quad + \mathbb{E}_{a\sim\pi_\theta}\left[(\nabla_\theta \log \pi_\theta(a|s))(\nabla_\theta \log \pi_\theta(a|s))^T \frac{\mathrm{Var}_k(\log \pi^k(a|s))}{N}\right] \end{aligned} \tag{52}$$

Learned policy prior. Assuming π_{θ_p} to be exact, the only source of randomness in g_2 is the sampling of actions a: $\mathrm{Var}_{a\sim\pi_\theta}(g_2) = \mathrm{Var}_{a\sim\pi_\theta}(\nabla_\theta \log \pi_\theta(a|s)\mathbb{E}_k[\log \pi^k(a|s)])$.

Which is identical to the first term in $\mathrm{Var}_{\substack{k \\ a\sim\pi_\theta}}(g_1(a))$. Then it follows that, under the condition that $N \ll K$ and that π^k are not all identical, the variance of gradient g_2 will be strictly lower than g_1.