Efficient Gaussian Processes for Model-based Online Planning

Hu Hanyang
Supervisor: Jonathan Scarlett

Department of Mathematics
National University of Singapore
Mathematics Capstone Project for Semester 2, AY2024/2025

Report Summary:

This report explores the integration of scalable Gaussian processes (GPs) into model-based online planning, focusing on balancing computational feasibility with non-parametric flexibility. Specifically, we investigate whether GP dynamics models can enhance the sample efficiency of model-based planning methods with less compromise in runtime. Our experiments across simulated continuous control tasks evaluate various empirical aspects of GP-based planning, including the impact of kernel selection (e.g., RBF v.s. Matérn) and the efficiency of variational conditioning and deep kernel learning (DKL). While our results suggest that Matérn kernels may improve contact dynamics modeling and DKL could reduce computational overhead (albeit trading off sample efficiency), challenges persist in environments with complex dynamics requiring more inducing points. Our study provides preliminary insights into practical GP-MBRL integration, with recommendations for future extensions in adaptive inducing point allocations, uncertainty quantification, and kernel selection for domain-specific applications.

Statement of Contributions:

- We propose a parallelizable framework combining MLP base models with GP corrections (Section 3.1), leveraging the model capacity of MLPs and the flexibility (as well as sample efficiency) of GPs.
- We suggest decoupling GP training and inference (Section 3.2), which contributes to the reduced total runtime and allows flexible choice of inference methods (e.g., LKI-based correction in Section 5.2.1 and dynamical local projection in Section 5.2.2).
- The computer program that generates the results demonstrated in Sections 3–5 are available online at `https://github.com/hanyang-hu/gp-mbrl`, which includes the implementation of GP training with subsampling and variational conditioning (Section 3.2), DKL with parameter sharing (Section 3.3), LKI-based correction (Section 5.2.1), dynamical local projection (Section 5.2.2), etc. The components of MLP model training and model predictive path integral (MPPI) are modified from the official implementation of TD-MPC [33].
- By re-ordering the H-step value expansions from ([96], Theorem 1) and introducing additional technical assumptions, we establish Theorem 1 and provide relevant discussions on the model-based value error of TD-MPC in Appendix A, explaining why TD-MPC achieves strong performance despite relying on deterministic models, and demonstrating that the dynamics gap depends jointly on the model bias (measured by the Wasserstein distance) and the underlying complexity of the MDP.

Contents

1 Introduction

Model-based reinforcement learning (MBRL), particularly model-based online planning, enables more flexible and sample-efficient reinforcement learning compared to the model-free alternatives [15, 83]. Meanwhile, Gaussian processes (GPs) may complement this framework by providing expressive, non-parametric modeling—particularly when compared to multi-layer perceptrons (MLPs). This synergy motivates the growing interest in leveraging GP-based dynamics models within model-based planning (e.g., [8, 37]) in the context of micro-data or sample-efficient reinforcement learning [13]. However, the widespread adoption of GP-based planning remains hindered by two persistent challenges: (1) the computational complexity; and (2) the curse of dimensionality.

Consequently, existing works on GP-based planning, or GP-based MBRL, are usually restricted to low-dimensional state-action spaces and/or small-scale training datasets (e.g., [15, 20, 37, 83]). Even in these constrained settings, empirical benchmarks demonstrate that GP-based MBRL can require computational times spanning multiple days [83].

Recent advances in scalable Gaussian process regressions (e.g., [55, 82, 93]) and increases in computational power hold promise for revitalizing research into GP-based reinforcement learning. Notably, Bosch et al. [8] explored the flexibility of GPs in image-based planning, overcoming the curse of dimensionality by learning the GP dynamics model in the latent space encoded by neural networks. Nevertheless, most methods (e.g., [8, 57]) remain largely confined to simple benchmarks such as the inverted pendulum. We posit that this limitation stems from the stringent real-time requirements of iterative inference during planning. In such settings, the associated computational overhead severely restricts the practicality of GP-based planning in more diverse domains.

In this report, our goal is to close the runtime gap between the MLP and GP dynamics models to facilitate the real-time deployment of GP-based online planning to more diverse domains. Our method is based on the following design choices (Section 3):

- **Integration with TD-MPC** [33]: Mitigating the computational bottleneck of GP-based planning by requiring fewer model inferences in the iterative trajectory rollouts.
- **GP-based correction of MLP dynamics**: Using the MLP as a base model and training GPs to correct the MLP predictions (possibly in parallel). This hybrid approach potentially outperforms standalone GPs and uncorrected MLPs.
- **Decoupled training-inference pipeline**: Separating offline hyperparameter optimization from online inference to minimize the training overhead of stochastic variational Gaussian processes (SVGPs) [47, 76] while preserving constant-time inference.
- **Integration with DKL** [92]: Balancing computational tractability and data efficiency by combining MLP feature extractors with GP modeling.

We evaluate different variants of our method (namely, GP-TD-MPC) on several OpenAI Gymnasium environments [78] and investigate their respective behaviors (Section 4).

2 Background

2.1 Reinforcement Learning and Markov Decision Processes

Consider an agent interacting with an environment. At each step, the agent (partially) observes the environment, takes an action, and then perceives feedback in the form of a new observation and a numerical reward signal. In reinforcement learning (RL), the agent improves its behavior over time via trial-and-error, seeking to maximize the cumulative sum of reward signals [74]. In this regard, Markov decision processes (MDPs) serve as a suitable formal framework, effectively representing essential features of an RL problem such as transitions of states, uncertainty in environments, presence of goals, etc.

An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, M, r, \rho_0)$ with state-space $\mathcal{S}$, action-space $\mathcal{A}$, transition probability distribution $s_{t+1} \sim M(\cdot|s_t, a_t)$, reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and initial state distribution $s_0 \sim \rho_0(\cdot)$.[1] The agent selects actions according to its policy π, which can be a deterministic function $\pi : \mathcal{S} \to \mathcal{A}$ or a probabilistic distribution $a_t \sim \pi(\cdot \,|\, s_t)$. The MDP's objective is to choose a policy that maximizes the expected return (i.e., some cumulative function of the random rewards).

Specifically, we consider the infinite-horizon discounted return

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \tag{2.1.1}$$

where $\gamma \in (0, 1)$ is the discount factor and $\tau = (s_0, a_0, s_1, a_1, \ldots)$ is a trajectory sampled by the agent acting in the environment. The optimal policy π^* could be defined by

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau \sim P^{\pi}(\cdot)}[R(\tau)] = \arg\max_{\pi} \mathbb{E}_{\tau \sim P^{\pi}(\cdot)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \tag{2.1.2}$$

where $P^{\pi}(\cdot)$ denotes the distribution of trajectory $\tau = (s_0, a_0, s_1, a_0, \ldots)$ sampled according to the initial state distribution $s_0 \sim \rho_0(\cdot)$, the policy $a_t = \pi(s_t)$ for deterministic policies (resp. $a_t \sim \pi(\cdot \,|\, a_t)$ for stochastic policies), and the dynamics $s_{t+1} \sim M(\cdot|s_t, a_t)$.

In practice, maximizing the objective in (2.1.2) is challenging since we usually only have access to a set of sampled trajectories with finite length collected from a (possibly non-optimal) behavior policy. In this case, it is often useful to introduce the notion of value functions, such as the action-value function $Q^{\pi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ w.r.t. a policy π given by

$$Q^{\pi}(s, a) = \mathbb{E}_{\tau \sim P^{\pi}(\cdot)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,\Big|\, s_0 = s, a_0 = a \right]. \tag{2.1.3}$$

Notice that $Q^{\pi}(s, a)$ equals the expected return $\mathbb{E}_{\tau}[R(\tau)]$ when the agent starts in state s, takes an arbitrary action a, and then continues to act according to the specified policy π.

[1] We may also use s_0 to denote the known initial state, the MDP is then defined by $\mathcal{M} = (\mathcal{S}, \mathcal{A}, M, r, s_0)$.

Assume we can find the optimal value function Q^* without knowing the corresponding optimal policy π^*, then the optimal policy could be extracted from Q^* greedily as $\pi(s) = \arg\max_{a\in\mathcal{A}} Q_\theta(s,a)$. This is the idea behind the classical Q-learning algorithm [86].

Suppose π is a deterministic policy, then the action-value function Q^π satisfies

$$Q^\pi(s,a) = r(s,a) + \gamma \cdot \mathbb{E}_{s'\sim M(\cdot|s,a)}[Q^\pi(s', \pi(s'))] \tag{2.1.4}$$

which is known as the Bellman equation. We also have the Bellman optimality equation

$$\begin{aligned} Q^*(s,a) &= r(s,a) + \gamma \cdot \mathbb{E}_{s'\sim M(\cdot|s,a)}[Q^*(s', \pi^*(s'))] \\ &= r(s,a) + \gamma \cdot \mathbb{E}_{s'\sim M(\cdot|s,a)}[\max_{a\in\mathcal{A}} Q^*(s',a))]. \end{aligned} \tag{2.1.5}$$

The Bellman equations allow us to train a parameterized model Q_θ to approximate Q^π given a policy π (or Q^*, which only requires solving $\max_{a\in\mathcal{A}} Q^*(s',a))$) by minimizing the mean-squared Bellman error (MSBE)

$$L(\theta, \mathcal{D}) = \mathbb{E}_{(s_t,a_t,s_{t+1},r_t)\sim\mathcal{D}}[(Q_\theta(s_t,a_t) - y_t)^2] \tag{2.1.6}$$

where $\mathcal{D}$ is a dataset sampled by a behavior policy and $y_t = r_t + \gamma Q_\theta(s_{t+1}, \pi(s_{t+1}))$ is the temporal-difference (TD) target. Intuitively, when $Q_\theta = Q^\pi$ and the behavior policy is π, the loss $L(\theta)$ is minimized according to the Bellman equations (2.1.4) and (2.1.5).

However, this vanilla approach has two limitations: (1) It is unstable for neural network approximators in practice; (2) It is hard to solve $\arg\max_{a\in\mathcal{A}} Q_\theta(s,a)$ over a continuous action space. To address the issue of unstable training, Mnih et al. proposed Deep Q-Networks (DQN) [49], which randomly samples batches of transitions $\{(s_t, a_t, s_{t+1}, r_t)\}$ from an experience replay buffer $\mathcal{D}$ to avoid issues of data correlation and non-stationary distributions. Furthermore, DQN computes the TD target

$$y_t = r_t + \gamma Q_{\theta_{\text{targ}}}(s_{t+1}, \pi(s_{t+1})) \tag{2.1.7}$$

by a target Q-network $Q_{\theta_{\text{targ}}}$, instead of the original Q_θ, to stabilize the minimization of the MSBE (2.1.6). After each optimization step, the parameters of the target Q-network θ_{targ} are updated by Polyak averaging with $\rho \in (0,1)$

$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta. \tag{2.1.8}$$

In addition, Lillicrap et al. proposed Deep Deterministic Policy Gradient (DDPG) [46], extending deep Q-learning to continuous action space by training a deterministic policy π_θ that approximates $\arg\max_{a\in\mathcal{A}} Q_\theta(s,a)$. Specifically, the policy parameters are learned by maximizing the following objective

$$\max_\theta \mathbb{E}_{s\sim\mathcal{D}}[Q_{\theta_{\text{targ}}}(s, \pi_\theta(s))] \tag{2.1.9}$$

via gradient ascent (where θ_{targ} is fixed in each optimization step). Notice that the gradient of the objective in (2.1.9) w.r.t. the policy parameters approximates the gradient of the objective in (2.1.2), as formally described in the deterministic policy gradient theorem [67].

2.2 Model-based Reinforcement Learning

All the aforementioned RL methods belong to the model-free family, where a dynamics model of state transitions is not explicitly learned. In contrast, model-based reinforcement learning (MBRL) methods learn a dynamics model to achieve better sample efficiency. Most MBRL methods can be classified into three categories: (1) Dyna-style algorithms, where model-free methods are trained on imaginary datasets generated by the dynamics model (e.g., Dyna [73]); (2) Policy search algorithms, where the derivatives of dynamics models are exploited (e.g., PILCO [20]); and (3) Online planning algorithms, where the dynamics model is directly used in model predictive control (e.g., PE-TS [15]). In this report, we focus on online planning algorithms due to their flexibility during deployment.

At each step t, model predictive control (MPC) usually considers the following trajectory optimization problem with a fixed horizon H

$$\max_{a_{t:t+H} \in \mathcal{A}^H} \mathbb{E}\left[\sum_{i=0}^{H} \gamma^i R_\theta(s_{t+i}, a_{t+i})\right] \tag{2.2.1}$$

where the trajectories are sampled by the learned dynamics model $s' \sim M_\theta(\cdot|s,a)$, whilst the reward function could be either given or learned. In this report, we consider a learned reward model $R_\theta(s,a)$. In response to the observation s_t, the agent takes the first action a_t^* of the optimal action sequence $a_{t:t+H}^*$, i.e.

$$\pi_{\text{MPC}}(s_t) = \arg\max_{a_t} \max_{a_{t+1:t+H}} \mathbb{E}\left[\sum_{i=0}^{H} \gamma^i R_\theta(s_{t+i}, a_{t+i})\right]. \tag{2.2.2}$$

Common choices of optimization algorithms for MPC include the cross-entropy method (CEM) [9] and model predictive path integral (MPPI) [90]. Both CEM and MPPI are population-based methods that iteratively update parameters for independent normal distributions (i.e., with diagonal covariance) that generate the new candidate solutions.

The primary difference between CEM and MPPI lies in the update rules for μ and Σ (Algorithm 1 Line 11). For CEM, the new mean and (diagonal) covariance parameters are the mean and (diagonal) covariance of the elite samples $\{\Gamma_k^*\}_{k=1}^K$ respectively, i.e.

$$\mu^j = \frac{\sum_{k=1}^K \Gamma_k^*}{K} \qquad \text{and} \qquad \sigma^j = \sqrt{\frac{(\Gamma_k^* - \mu^j)^2}{K}} \tag{2.2.3}$$

where the top-K estimated returns $\{\phi_{\Gamma_k^*}\}_{k=1}^K$ are not utilized. In contrast, for MPPI, each elite sample Γ_k^* is weighted by $\Omega_k = \exp(\tau\phi_{\Gamma_k^*})$ with a temperature hyperparameter τ. Specifically, the update rule of MPPI is

$$\mu^j = \frac{\sum_{k=1}^K \Omega_k \Gamma_k^*}{\sum_{k=1}^K \Omega_k} \qquad \text{and} \qquad \sigma^j = \max\left(\sqrt{\frac{\sum_{k=1}^K \Omega_k (\Gamma_k^* - \mu^j)^2}{\sum_{k=1}^K \Omega_k}}, \epsilon\right). \tag{2.2.4}$$

where $\epsilon > 0$ is a linearly decayed constant introduced to prevent the algorithm from becoming trapped in potentially suboptimal local solutions [33].

Algorithm 1 Model Predictive Control (MPC)

1: **Input:** Number of iterations J, population size N, number of elite samples K, rollout horizon H, initial distribution parameters μ^0, Σ^0, (learned) dynamics model M_θ, (learned) reward model R_θ, current state s_t.
2: **for** each iteration $i = 1, 2, \ldots, J$ **do**
3: Sample N action sequences of length H from $\mathcal{N}(\mu^{j-1}, \Sigma^{j-1})$.
4: **for** all N sequences $\Gamma = (a_t, a_{t+1}, \ldots, a_{t+H})$ **do**
5: **for** step $j = 0, 1, \ldots, H-1$ **do** ▷ Estimate trajectory return ϕ_Γ
6: Update $\phi_\Gamma = \phi_\Gamma + \gamma^t R_\theta(s_{t+j}, a_{t+j})$. ▷ Initially setting $\phi_\Gamma = 0$
7: Predict $s_{t+j+1} \sim M_\theta(s_{t+j}, a_{t+j})$.
8: **end for**
9: **end for**
10: Select the elite samples $\{\Gamma_k^*\}_{k=1}^K$ corresponding to the top-K returns $\{\phi_{\Gamma_k^*}\}_{k=1}^K$.
11: Update parameters μ^j, σ^j for the next iteration based on $\{\Gamma_k^*\}_{k=1}^K$ and $\{\phi_{\Gamma_k^*}\}_{k=1}^K$.
12: **end for**
13: **Output:** $(a_t^*, a_{t+1}^*, \ldots, a_{t+H}^*) \sim \mathcal{N}(\mu^J, \Sigma^J)$

Since the MPC optimization procedure is executed at each sampling time, strategies that reduce the number of iterations J required for convergence are crucial for real-time applications. For example, one may shift the mean parameter μ_{prev}^J obtained from the previous step and reuse it as μ^0 in the current step [2, 33], assuming that the optimal trajectory evolves smoothly over time. Furthermore, in LOOP [66] and TD-MPC [33], a policy prior π_θ is learned so that the trajectories generated by π_θ are mixed with the sampled trajectories (in Algorithm 1 Line 3) to guide the optimization procedure. Specifically, TD-MPC [33] learns an action-value function Q_θ from the replay buffer and extracts the policy π_θ via DDPG [46] as described in Section 2.1.

In addition to the number of iterations J, the planning horizon H is also a crucial factor in MPC's efficiency. Typically, a planning horizon from 20 to 40 works best for many MBRL methods [83]. However, TD-MPC [33] requires a significantly shorter planning horizon (e.g., $H = 5$) by augmenting the action-value function Q_θ into the optimization objective as a terminal value

$$\max_{a_{t:t+H} \in \mathcal{A}^H} \mathbb{E}\left[\sum_{i=0}^{H-1} \gamma^i \hat{r}(s_{t+i}, a_{t+i}) + \gamma^H Q_\theta(s_H, a_H)\right] \tag{2.2.5}$$

in contrast to the optimization objective in (2.2.1). Intuitively, as TD learning approximates the global optimal solution, MPC only needs to focus on yielding the optimal local

behaviors, requiring shorter prediction horizons. In the meantime, the compounding errors of the dynamics model prediction could also be alleviated with a smaller H.

To justify these insights, we aim to bound the H-step model-based value error of TD-MPC. This error is important as it partially governs the suboptimality of model-based planning methods [66, 84] (with another factor being the suboptimality of MPC optimizations occurred in each timestep). Intuitively, when the model-based value estimation has large errors, the planner may incorrectly evaluate the expected performance of candidate action sequences (or candidate policies), resulting in suboptimal behaviors.

In particular, we consider the Wasserstein distance $W(\cdot, \cdot)$ instead of the total variation distance $d_{\mathrm{TV}}(\cdot, \cdot)$ to define the dynamics model error ϵ_m, as TD-MPC uses a deterministic dynamics model (see discussions in Appendix A.1). This approach may require additional assumptions on the Lipschitz continuity of components in the original MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, M, r, \rho_0)$, in contrast to the analysis of compounding errors based on the Markov chain TVD bound [39, 66]. Nevertheless, such assumptions are not uncommon in the theoretical analysis of MBRL methods (e.g., [19, 70]). For example, predictions of MLPs with Lipschitz continuous activation functions (e.g., $\mathrm{ReLU}(\cdot)$), or Gaussian processes with Lipschitz continuous kernels, would also be Lipschitz continuous [19].

Assumption 1 (Bounded Value Function)**.** Suppose there exists a constant $V_{\max} < \infty$ such that $|V^\pi(s)| \leq V_{\max}$ for all $s \in \mathcal{S}$ and $\pi \in \Pi$.

Assumption 2 (Lipschitz Continuity)**.** The reward function r is L_R-Lipschitz continuous, and any deterministic policy $\pi \in \Pi$ is L_π-Lipschitz continuous. Furthermore, the stochastic dynamics M is L_M-Lipschitz continuous in the sense that

$$D_{\mathrm{TV}}(M(\cdot|s_1, a_1), M(\cdot|s_2, a_2)) \leq L_M \|(s_1, a_1) - (s_2, a_2)\|_2 \tag{2.2.6}$$

for all $s_1, s_2 \in \mathcal{S}$ and $a_1, a_2 \in \mathcal{A}$, where $D_{\mathrm{TV}}(\cdot, \cdot)$ denotes the total variation distance [79].

Remark. Assumption 1 should be a slightly weaker condition compared to assuming $r(s, a) \in [0, R_{\max}]$ and $V_{\max} = \max_s V^*(s)$ for all $s \in \mathcal{S}$ (where V^* denotes the optimal value function) similar to those in Sikchi et al. [66]. For Assumption 2, when we are only interested in policies determined by fixed action sequences (as in the MPC setting) instead of being a deterministic function, we may consider $L_\pi = 0$.

Based on these assumptions, we can derive an upper bound of the model-based value estimation error, which decomposes into a dynamics gap and a return estimation gap.

Theorem 1 (H-step Model-based Value Error)**.** *Given a policy $\pi \in \Pi$. Suppose $\hat{M}$ is a (deterministic) dynamics model such that $\max_{s,a} W(M(\cdot|s, a), \hat{M}(\cdot|s, a)) \leq \epsilon_m$, $\hat{r}$ is an approximate reward model such that $\max_{s,a} |r(s, a) - \hat{r}(s, a)| \leq \epsilon_r$, and $\hat{Q}^\pi$ is an approximate*

action-value function such that $\max_{s,a} |Q^\pi(s,a) - \hat{Q}^\pi(s,a)| \leq \epsilon_q$. *Let* V^π *denote the ground-truth value function and* $\hat{V}^\pi$ *denote the model-based value estimation*

$$\hat{V}^\pi(s) = \mathbb{E}_{\hat{\tau} \sim \hat{P}^\pi(\cdot|s)} \left[\sum_{t=0}^{H-1} \gamma^t \hat{r}(s_t, a_t) + \gamma^H \hat{Q}(s_H, a_H) \right] \tag{2.2.7}$$

where $\hat{\tau} = (s_0, \pi(s), \ldots, s_H, \pi(s_H))$ *is a trajectory sampled by* $\hat{M}$ *and* π *starting at the initial state* $s_0 = s$ *(i.e.,* $\hat{\tau} \sim \hat{P}^\pi(\cdot|s)$*), then there is a constant factor* $K_{\mathcal{M}}$ *such that*

$$\left| V^\pi(s) - \hat{V}^\pi(s) \right| \leq \underbrace{K_{\mathcal{M}} \frac{\gamma - \gamma^{H+1}}{1-\gamma} \epsilon_m}_{\textit{dynamics gap}} + \underbrace{\frac{1-\gamma^H}{1-\gamma} \epsilon_r + \gamma^H \epsilon_q}_{\textit{return estimation gap}} \tag{2.2.8}$$

for any initial state $s \in \mathcal{S}$, *where* $K_{\mathcal{M}}$ *satisfies* $K_{\mathcal{M}} \leq (L_R + 2\gamma V_{\max} L_M)\sqrt{1 + L_\pi^2}$.

Proof. We defer the proof to Appendix A.2. □

Theorem 1 decomposes the value error bound into the sum of the dynamics gap and the return estimation gap. Interestingly, the dynamics gap is influenced not only by the dynamics model error ϵ_m but also by the complexity of the original MDP, which is determined by the Lipschitz constants of the ground-truth stochastic dynamics M, the reward function r, and the policy class Π. On the other hand, when the model $\hat{M}$ closely approximates the ground-truth dynamics and has a small Lipschitz constant, the compounding error of multi-step predictions is also expected to be small (see Appendix A.3).

We may also observe that the planning horizon H introduces a trade-off between the dynamics gap and the return estimation gap [66]. When $H = 0$, the learned dynamics model $\hat{M}$ and the reward model $\hat{r}$ have no effect on the RHS of (2.2.8), as TD-MPC would become model-free and the upper bound is simply the value estimation error ϵ_q, which could be large in the low-data regime as evidenced by the superiority of MBRL methods in such cases (e.g., [15, 39]). As H increases, the model error compounds, resulting in a larger dynamics gap, while the effects of ϵ_q decay by a factor of γ^H. Furthermore, this analysis also applies to MPC without the TD learning component. Consider $Q_\theta \equiv 0$, then the value error becomes $\epsilon_q = \max_{s,a} |Q^\pi(s,a)|$, which would potentially be much larger compared to $\max_{s,a} |Q^\pi(s,a) - Q_\theta(s,a)|$ with a learned action-value function, justifying the combination of temporal-difference learning and model-based planning in (2.2.5).

Remark. It should be possible to leverage the result of Theorem 1 (potentially, with some minor modifications) to establish a performance lower bound for TD-MPC, following steps similar to those of Sikchi et al. [66]. This lower bound would be based on the RHS of (2.2.8), as well as a suboptimality gap ϵ_p incurred during the MPC optimization w.r.t. (2.2.5). However, we do not elaborate on this as it may not yield additional insights.

2.3 Gaussian Processes for Machine Learning

The Gaussian process (GP) [61] is a compelling alternative to neural networks for machine learning. It provides a mechanism to directly quantify the uncertainty and is data-efficient by its Bayesian non-parametric nature, making it especially suitable in the case of learning robot controllers with a handful of trials since real-world data are usually expensive to gather [20]. However, similar to other Bayesian methods, the standard GPs are computation-heavy, hence requiring specific measures to achieve scalability, including structural assumptions on kernel functions/matrices [23, 93], kernel approximations [59, 89], etc. In this subsection, we briefly overview the mathematical formulation of GPs and the training and inference of exact GP models for regression tasks.

A Gaussian process is a function prior $\hat{f} \sim \mathcal{GP}(m(\cdot), k(\cdot,\cdot))$ specified by the mean function $m(\cdot)$ and the kernel function $k(\cdot,\cdot)$ (i.e., the function-space view) [61]. Conditioned on the input $X = (\mathbf{x}_1, \dots, \mathbf{x}_n) \in \mathcal{X}^n$ where $\mathcal{X} \subseteq \mathbb{R}^D$ and observations $\mathbf{y} = (y_1, \dots, y_n)$, the value of the function $\hat{f}(\mathbf{x})$ at any given location $\mathbf{x} \in \mathcal{X}$ is normally distributed with mean and variance in closed-form expressions of $m(\cdot)$ and $k(\cdot,\cdot)$.

Specifically, suppose $y = f(x) + \epsilon$, $f \sim \mathcal{GP}(0, k(\cdot,\cdot))$ (i.e., $m(\cdot) \equiv 0$), and $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$. Given the training data $\mathcal{D} = (X, \mathbf{y})$ and test point $\mathbf{x}^*$, let K_{XX} be the Gram matrix of X defined by $[K_{XX}]_{i,j} := k(\mathbf{x}_i, \mathbf{x}_j)$ and $\mathbf{k}_{X\mathbf{x}^*}$ be an n-dimensional vector defined by $[\mathbf{k}_{X\mathbf{x}^*}]_i := k(\mathbf{x}_i, \mathbf{x}^*)$, put $\hat{K}_{XX} = K_{XX} + \sigma_\epsilon^2 \mathbf{I}$, then the predictive posterior distribution of $f(\mathbf{x}^*)$ is a normal distribution with:

$$\begin{aligned} \mathbb{E}[f(\mathbf{x}^*) \,|\, X, \mathbf{y}, \mathbf{x}^*] &= \mathbf{k}_{X\mathbf{x}^*}^\top \hat{K}_{XX}^{-1} \mathbf{y} \\ \mathrm{Var}[f(\mathbf{x}^*) \,|\, X, \mathbf{y}, \mathbf{x}^*] &= k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}_{X\mathbf{x}^*}^\top \hat{K}_{XX}^{-1} \mathbf{k}_{X\mathbf{x}^*} \end{aligned} \tag{2.3.1}$$

When it comes to an arbitrary mean function $m(\cdot)$, we could equivalently apply equations (2.3.1) on the transformed data $(\mathbf{x}, y) \mapsto (\mathbf{x}, y - m(\mathbf{x}))$, then add $m(\mathbf{x})$ back to the posterior mean prediction. Notably, the posterior variance is independent of the choice of $m(\cdot)$.

The most popular choice of kernel functions are the stationary kernel functions such as the Radial Basis Function (RBF) kernel or the Matérn kernel [61]:

$$\begin{aligned} k_l^{\text{RBF}}(\mathbf{x}_1, \mathbf{x}_2) &= \exp\left(-\sum_{i=1}^{D} \frac{(x_{1,i} - x_{2,i})^2}{2[l]_i^2} \right) \\ k_{l,\nu}^{\text{Matérn}}(\mathbf{x}_1, \mathbf{x}_2) &= \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \sum_{i=1}^{D} \frac{|x_{1,i} - x_{2,i}|}{[l]_i} \right)^\nu K_\nu \left(\sqrt{2\nu} \sum_{i=1}^{D} \frac{|x_{1,i} - x_{2,i}|}{[l]_i} \right) \end{aligned} \tag{2.3.2}$$

where $l \in \mathbb{R}_{>0}^D$ is the lengthscale hyperparameter indicating the importance of each dimension, $\Gamma(\cdot)$ is the Gamma function, and $K_\nu(\cdot)$ is the modified Bessel function of the second kind. For the Matérn kernel, $\nu > 0$ is a hyperparameter that indicates the differentiability of the function sampled from the corresponding GP. As $\nu \to \infty$, the Matérn kernel approximates the RBF kernel. In practice, they are often wrapped by a scale kernel such that $K_{\text{scaled}} = \sigma_f^2 K_{\text{orig}}$ where σ_f^2 is the scale parameter.

These kernels are stationary in the sense that, for any $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$, the kernel value $k(\mathbf{x}_1, \mathbf{x}_2)$ only depends on $\mathbf{x}_1 - \mathbf{x}_2$, independent of their exact positions (hence we could write $k(\mathbf{x}_1 - \mathbf{x}_2) = k(\mathbf{x}_1, \mathbf{x}_2)$). In general, Bochner's theorem [62] guarantees that any stationary kernel $k(\cdot)$ must be the Fourier transform of a non-negative measure $p(\cdot)$, i.e.

$$k(\mathbf{x}_1 - \mathbf{x}_2) = \int p(\omega) e^{j\omega^\top (\mathbf{x}_1 - \mathbf{x}_2)} \, d\omega = \mathbb{E}_{\omega \sim p}[\xi_\omega(\mathbf{x}_1) \xi_\omega(\mathbf{x}_2)^*] \tag{2.3.3}$$

where $\xi_\omega(\mathbf{x}) = e^{j\omega^\top \mathbf{x}}$. Since $k(\cdot)$ is symmetric, the spectral density $p(\cdot)$ in the frequency domain (2.3.3) must also be symmetric. Therefore, if we approximate the spectral density $p(\cdot)$ by a symmetrically weighted mixture of Q square-exponential functions, then the corresponding kernel, known as the spectral mixture (SM) kernel [91], takes the form

$$k^{\mathrm{SM}}_{l,w,\mu}(\mathbf{x}_1, \mathbf{x}_2) = \sum_{q=1}^{Q} w_q \exp\left(- \sum_{i=1}^{D} \frac{(x_{1,i} - x_{2,i})^2}{2[l]^2_{q,i}} \right) \cos\left(\sum_{i=1}^{D} [\mu]_{q,i} (x_{1,i} - x_{2,i}) \right) \tag{2.3.4}$$

which is particularly effective for extrapolation due to its quasi-periodic structure [91, 92]. Moreover, by the universal function approximation property of the mixtures of Gaussians, SM kernels can approximate any stationary kernel arbitrarily well given a sufficiently large number of mixture components Q, which mitigates the kernel selection problem [53]. Nevertheless, the SM kernel typically requires greater computational resources (e.g., memory and processing time).

The stationary covariance structures generally assumed by these kernels may not adequately capture variability in non-stationary data [6]. Furthermore, these kernels may suffer from the curse of dimensionality (CoD) as the distance between points in high-dimensional space tends to increase [43]. Consequently, in non-stationary and/or high-dimensional settings, we may introduce a neural network feature extractor as kernel hyperparameters, known as deep kernel learning (DKL) [92], which enjoys the flexibilities of both neural network architectures and non-parametric kernel methods.

Training a GP model involves maximizing the marginal log-likelihood (MLL) w.r.t. the kernel hyperparameters θ (e.g., the lengthscales l, the DKL parameters, etc.)

$$\mathcal{L} = \log p(\mathbf{y} \mid X, \theta) \propto - \underbrace{\mathbf{y}^\top \hat{K}_{XX}^{-1} \mathbf{y}}_{\text{error}} - \underbrace{\log |\hat{K}_{XX}|}_{\text{complexity}}. \tag{2.3.5}$$

Furthermore, the gradient of $\mathcal{L}$ over θ is given by

$$\frac{\partial \mathcal{L}}{\partial \theta} \propto \mathbf{y}^\top \hat{K}_{XX}^{-1} \frac{\partial \hat{K}_{XX}}{\partial \theta} \hat{K}_{XX}^{-1} \mathbf{y} - \operatorname{tr}\left(\hat{K}_{XX}^{-1} \frac{\partial \hat{K}_{XX}}{\partial \theta} \right) \tag{2.3.6}$$

since $\hat{K}_{XX}$ is symmetric positive definite [61].

From equations (2.3.1), (2.3.5), and (2.3.6), the training and inference of GPs involve computing the matrix solves $\hat{K}_{XX}^{-1}\mathbf{y}$ and $\hat{K}_{XX}^{-1}\mathbf{k}_{X\mathbf{x}^*}$, the determinant term $|\hat{K}_{XX}|$, and the trace term $\text{tr}\left(\hat{K}_{XX}^{-1}\frac{\partial \hat{K}_{XX}}{\partial \theta}\right)$, which are the main computational bottlenecks. A typical method used for computing these terms is the Cholesky decomposition [30], which factorizes the symmetric positive definite matrix $\hat{K}_{XX}$ into $LL^\top$ where L is a lower triangular matrix so that computing matrix solve and log determinants take $O(n^2)$ and $O(n)$ times respectively. However, directly applying Cholesky decomposition is inefficient for large-scale applications since it takes $O(n^3)$ time for computation, $O(n^2)$ space for storage, and is less amenable to GPU acceleration.

Common approaches to enhance GP's scalability include black-box MVM-based methods [28, 82], subsampling [35, 98], kernel matrix approximation (e.g., Fourier features [52, 59], Nyström approximation [89], Lanczos decomposition [55], etc.), structured kernel interpolation (SKI) [29, 69, 93], and stochastic variational Gaussian processes (SVGP) [47, 76]. We provide an overview of some scalable GP regression methods in Appendix B, with connections to our focus on model-based planning when applicable.

2.4 Gaussian Processes for Reinforcement Learning

The application of Gaussian processes in reinforcement learning has been extensively studied, spanning areas such as temporal-difference learning (e.g., GP-SARSA [26, 48]) and model-based policy search (e.g., PILCO [21], Black-DROPS [12, 13], and BAGEL [75]). The common practice in model-based approaches is to train $\dim(\mathcal{S})$ independent GPs, each takes $(s, a) \in \mathcal{S} \times \mathcal{A}$ as input and predicts one entry of the next state $s' \in \mathcal{S}$. This has often been successful for low-dimensional robotics tasks within limited interaction with the real machine. However, many of these methods fail to scale in high-dimensional environments and are extremely time-consuming to train on large datasets [15, 83].

The scalability issue of GPs would be further amplified in GP-based planning, where the time constraint is more stringent for real-world applications. To enhance computation efficiency, previous approaches mainly employ sparse GPs and approximate uncertainty propagation [37, 81]. To alleviate the curse of dimensionality (e.g., for pixel-based planning), the deep latent Gaussian process dynamics (DLGPD) approach [8] trains the latent dynamics GP model jointly with a neural network auto-encoder, optimizing the evidence lower bound (ELBO) composed of reconstruction loss terms and a KL regularization term similar to the variational auto-encoder [41] and several other MBRL methods based on pixel reconstruction (e.g., [31, 32]).

Another common issue for GP-based RL methods is numerical stability, which becomes particularly pronounced when combining GPs with neural network encoders. To address

this, DLGPD [8] incorporates a signal-to-noise ratio (SNR) penalty into the loss function

$$\mathcal{L}_{\text{SNR}} = \sum_{k=1}^{\dim(\mathcal{S})} \left(\frac{\log(\sigma_{f,k}/\sigma_{\epsilon,k})}{\log \tau} \right)^p \tag{2.4.1}$$

where τ, p are hyperparameters, $\sigma_{f,k}^2$ is the scale parameter and $\sigma_{\epsilon,k}^2$ is the noise variance of the k-th independent GP. This regularization term encourages the noise variance $\sigma_{\epsilon,k}^2$ to be appropriately large w.r.t. the scale $\sigma_{f,k}^2$, allowing more stable numerical solutions for $\hat{K}_{XX}^{-1} = (K_{XX} + \sigma_\epsilon^2 \mathbf{I})^{-1}$ required for GP training and inference described in Section 2.3.

3 Methods

3.1 GP-based TD-MPC

In this report, we adopt the Temporal Difference Model Predictive Control framework (TD-MPC) [33] as our baseline. As detailed in Section 2.2, TD-MPC combines short-term dynamics predictions (MPC) with long-term value estimations (TD learning), enabling efficient decision-making in complex environments. Notably, we have chosen to implement the encoder-free variant of TD-MPC. This decision is motivated by its improved training stability (particularly for GP models) and sample efficiency in the low-data regime. Although it exhibits a slight disadvantage compared to the original TD-MPC according to the referenced experiments [33], the trade-off in stability makes it a favorable choice for our implementation. In addition, we have noticed that incorporating the SNR penalty described in Section 2.4 is not necessary in this case.

Similar to TD-MPC, our agent iteratively trains the model using data collected from previous interactions while acquiring new data through online planning. The encoder-free variant of TD-MPC consists of the following MLP components parameterized by θ:

- **Dynamics Prediction:**
$$\hat{s}_{t+1} = M_\theta(s_t, a_t)$$
- **Reward Prediction:**
$$\hat{r}_t = R_\theta(s_t, a_t)$$
- **Action-Value Prediction:**
$$\hat{q}_t = Q_\theta(s_t, a_t) := \min\left\{ Q_\theta^{(1)}(s_t, a_t), Q_\theta^{(2)}(s_t, a_t) \right\}$$
- **Deterministic Policy:**
$$a_t = \pi_\theta(s_t)$$

where the action-value prediction is taken as the minimum of two learned Q-networks, a technique known as clipped double Q-learning [27], which helps mitigate the overestimation bias in action-value predictions.

During each iteration, TD-MPC minimizes a temporally weighted objective

$$\mathcal{L}(\theta;\Gamma) = \sum_{i=t}^{t+H} \lambda^{i-t} \mathcal{L}(\theta;\Gamma_i) \tag{3.1.1}$$

where $\lambda \in (0,1)$ is a discount factor (that may differ from γ) and $\Gamma \sim \mathcal{B}$ is a trajectory $(s_t, a_t, r_t, s_{t+1})_{t:t+H}$ sampled from a replay buffer. In particular, the prioritized experience replay (PER) [63] strategy is employed, which replays transitions with larger TD loss more frequently. Moreover, the single-step loss is given by

$$\begin{aligned}\mathcal{L}(\theta;\Gamma_i) = c_1 &\underbrace{\|R_\theta(s_i,a_i) - r_i\|_2^2}_{\text{reward loss}} \\ &+ c_2 \underbrace{\|Q_\theta(s_i,a_i) - (r_i + \gamma Q_{\theta_{\text{targ}}}(s_{i+1}, \pi(s_{i+1})))\|_2^2}_{\text{TD loss}} \\ &+ c_3 \underbrace{\|s_{i+1} - M_\theta(s_i,a_i)\|_2^2}_{\text{dynamics loss}}\end{aligned} \tag{3.1.2}$$

with hyperparameters $c_1, c_2, c_3 \in \mathbb{R}_{>0}$. Subsequently, the policy π_θ is learned by minimizing the following objective

$$\mathcal{L}_\pi(\theta;\Gamma) = -\sum_{i=t}^{t+H} \lambda^{i-t} Q_\theta(s_i, \pi_\theta(s_{i+1})) \tag{3.1.3}$$

which is a temporally weighted variant of DDPG [46] described in Section 2.1.

In this report, we explore the application of Gaussian processes in model-based planning by replacing the dynamics model M_θ and the reward model R_θ with GPs, aiming to further enhance the sample efficiency of MBRL methods while adhering to constraints for real-time applications. However, rather than directly replacing the MLP models with GPs, we treat M_θ and R_θ as "base models" and use GPs to learn the MLP model residuals. In particular, the i-th training target (i.e., the i-th entry of the $\mathbf{y}$ term in equation (2.3.5)) for the multi-output GP that jointly models the dynamics and reward is replaced by the residual $\mathbf{y}_i - f_\theta(\mathbf{x}_i) = (s_{i+1} - M_\theta(s_i,a_i), r_i - R_\theta(s_i,a_i))$ instead of the ground-truth $\mathbf{y}_i = (s_{i+1}, r_i)$, incorporating the prior assumption that the base model provides a reasonably accurate approximation (see Figure 1). During inference, the GP predictions will be added to the predictions of the base models, serving as a correction term as follows

$$\text{output}(\mathbf{x}^*) = f_\theta(\mathbf{x}^*) + \underbrace{\mathbf{k}_{X\mathbf{x}^*}^\top \hat{K}_{XX}^{-1}(\mathbf{y} - f_\theta(X))}_{\text{GP correction}} \tag{3.1.4}$$

where f_θ is the base model (i.e., the dynamics model M_θ and the reward model R_θ). Notice that this could be interpreted as treating the base model f_θ as the prior mean function, hence it should not affect the uncertainty quantification of GPs.

We refer to this combined approach, integrating GP models with TD-MPC, as GP-TD-MPC. By employing the MLP base model f_θ as the prior mean function, GP-TD-MPC distinguishes itself from similar GP-based planning methods that forward-propagate the predictive mean, such as GP-E (in PE-TS) [15] and DLGPD [8], which allows us to leverage the representational capacity of neural networks as well as the non-parametric flexibility of GPs. Notice that, in such settings, we do not exploit the GP's inherent capability for uncertainty quantification. However, constant-time probabilistic inference can always be resumed via caching [55] or pathwise conditioning [94, 95], particularly when employing SVGPs [47, 76], which we shall elaborate in Section 3.2 and Section 5.3.2.

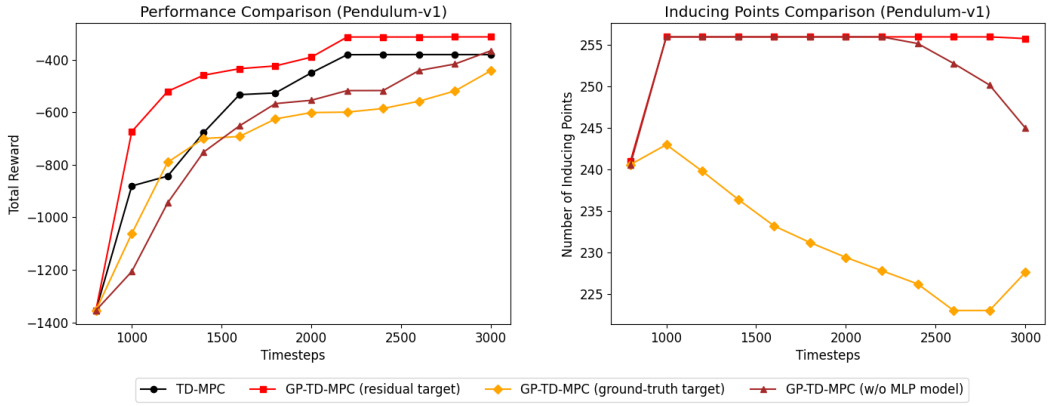


Figure 1: Comparison of the performance and number of inducing points (at most 256) determined by the pivoted Cholesky decomposition (as detailed in Section 3.2) for different design choices tested in the inverted pendulum swingup environment. For GP-TD-MPC, we use the standard RBF kernel. The results displayed are averaged over 5 repetitions. The following options are compared with the TD-MPC (no latent) baseline: (1) GP model trained with the residual targets $s_{i+1} - M_\theta(s_i, a_i)$ and $r_i - R_\theta(s_i, a_i)$, which correct the output of the base model; (2) GP model trained with the ground-truth targets s_{i+1} and r_i, which also correct the output of the base model; and (3) GP model trained with the ground-truth targets s_{i+1} and r_i, but directly make predictions without correcting the base model. Only option (1) outperforms the baseline.

3.2 Scalable Training and Inference

Notice that the training process of TD-MPC aforementioned cannot be directly applied to GP-based dynamics and reward models. The key limitation arises from the GP training loss in equation (2.3.5), which does not naturally decompose into a sum of independent terms suitable for mini-batch stochastic gradient descent as well as temporally weighted objectives. More importantly, as the size of the dataset increases, computing the MLL loss (2.3.5) and its derivative (2.3.6) may become computationally inefficient.

A common approach to address this issue is stochastic variational Gaussian processes (SVGP, see Appendix B.3), which approximates exact GPs by introducing a variational distribution $\mathbf{u} \sim q(\cdot)$ over pseudo targets evaluated at m inducing points Z, generating the latent functions via $p(\mathbf{f}|Z, \mathbf{u})$ [47, 76] so that the MLL in (2.3.5) can be bounded below, known as the evidence lower bound (ELBO), by applying Jensen's inequality

$$\log p(\mathbf{y}|X, \theta) \geq \mathbb{E}_{q(\mathbf{u})p(\mathbf{f}|\mathbf{u})}[\log p(\mathbf{y}|\mathbf{f})] - D_{\mathrm{KL}}(q(\mathbf{u})\|p(\mathbf{u})) \tag{3.2.1}$$

which admits stochastic gradients since the likelihood $p(\mathbf{y}|\mathbf{f}) = \prod_{i=1}^{n} p(\mathbf{y}_i|\mathbf{f}_i)$ usually factorizes over data instances.

Moreover, the GP posterior inference in (2.3.1) can be approximated by

$$\begin{aligned} \mathbb{E}[f(\mathbf{x}^*) \,|\, X, \mathbf{y}, \mathbf{x}^*] &\approx \mathbb{E}[f(\mathbf{x}^*) \,|\, Z, \mathbf{x}^*] = \mathbf{k}_{Z\mathbf{x}^*}^\top K_{ZZ}^{-1} \mathbf{m}_\mathbf{u} \\ \mathrm{Var}[f(\mathbf{x}^*) \,|\, X, \mathbf{y}, \mathbf{x}^*] &\approx \mathrm{Var}[f(\mathbf{x}^*) \,|\, Z, \mathbf{x}^*] \\ &= k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}_{Z\mathbf{x}^*}^\top K_{ZZ}^{-1}(K_{ZZ} - S_\mathbf{u}) K_{ZZ}^{-1} \mathbf{k}_{Z\mathbf{x}^*} \end{aligned} \tag{3.2.2}$$

when the variational distribution $q(\cdot)$ of the pseudo target $\mathbf{u}$ evaluated at Z is given by $\mathcal{N}(\mathbf{m}_\mathbf{u}, S_\mathbf{u})$. The GP-based correction in equation (3.1.4) can be modified correspondingly by replacing the (residual) training data $(X, \mathbf{y} - f_\theta(X))$ with the pseudo data $(Z, \mathbf{u})$ such that $\mathbf{u} \sim \mathcal{N}(\mathbf{m}_\mathbf{u}, S_\mathbf{u})$. We only consider the predictive mean of the corrected output, hence

$$\mathrm{output}(\mathbf{x}^*) = f_\theta(\mathbf{x}^*) + \underbrace{\mathbf{k}_{Z\mathbf{x}^*}^\top \hat{K}_{ZZ}^{-1} \mathbf{m}_\mathbf{u}}_{\text{SVGP correction}}. \tag{3.2.3}$$

The term $a = K_{ZZ}^{-1}\mathbf{m}_\mathbf{u}$ could be cached to achieve $O(m)$-time inference.

As a result, SVGPs not only enable more flexible training schemes but also has greater or comparable inference efficiency for applications in online planning compared to other approaches such as sparse GPs [64, 68], structured kernel interpolation (SKI) [29, 91], and caching [55, 82] (see Figure 2 for an empirical comparison). Nevertheless, it would introduce $O(m^2)$ additional learnable parameters, primarily from the covariance matrix $S_\mathbf{u}$ of the variational distribution $q(\cdot)$. This increases the computational requirement for training compared to standard GPs, which typically optimize $O(D)$ parameters only (e.g., the lengthscales $l \in \mathbb{R}_{>0}^D$).

Instead of using the ELBO loss (3.2.1) of SVGP, we employ the subsampling strategy for GP training as suggested by Hayashi et al. [35] and Zhao et al. [98], which minimizes

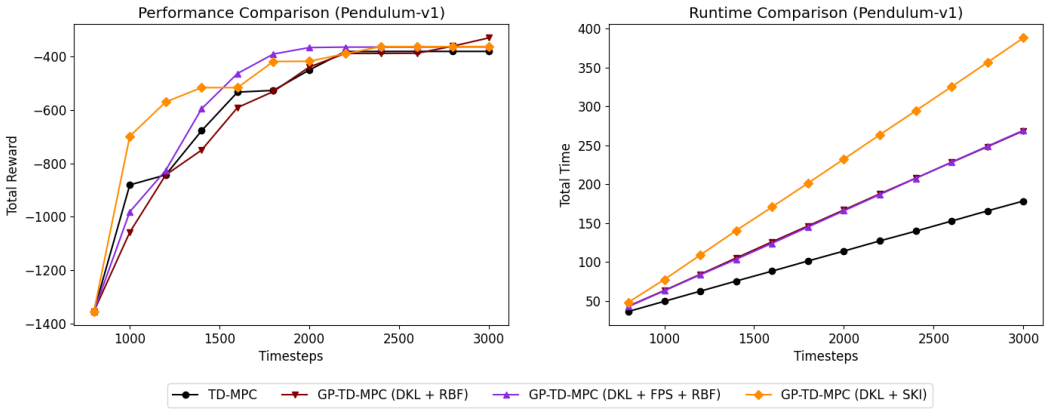


Figure 2: Comparison of the runtime in the Pendulum-v1 environment. The SVGP correction in equation (3.2.3) is at least as fast as cached exact/sparse GPs, hence we only compare it with local kernel interpolation (LKI), which is one of the key techniques used in structured kernel interpolation (SKI) [91]. (**Note.** For more details of LKI and SKI, see Section 5.2.1 and Appendix B.2). The LKI's mean prediction is $O(1)$-time with any number of inducing points. Nevertheless, its runtime is considerably longer than that of variational conditioning, suggesting that the interpolation step introduces additional overheads. Notice that since the training methods are similar, the runtime difference is mainly due to the different caching and inference speeds.

the MLL loss in equation (2.3.5) with a subsampled mini-batch of data[2], avoiding the need to introduce $O(m^2)$ additional parameters. Nevertheless, we could still enjoy the advantage of constant-time inference of SVGPs by considering the following parameterization used in online variational conditioning (OVC) [47] (see Appendix B.3).

$$\mathbf{c} = K_{ZX}\Sigma_{\mathbf{y}}^{-1}(\mathbf{y} - f_\theta(X)) \in \mathbb{R}^m, \qquad C = K_{ZX}\Sigma_{\mathbf{y}}^{-1}K_{XZ} \in \mathbb{R}^{m\times m} \tag{3.2.4}$$

where $\Sigma_{\mathbf{y}} = \sigma_\epsilon^2\mathbf{I}$ denotes the covariance of the likelihood $p(\mathbf{y}\,|\,\mathbf{f})$ and we condition on the MLP model residuals $\mathbf{y} - f_\theta(X)$ as described in Section 3.1. The optimal variational distribution $q = \mathcal{N}(\mathbf{m_u}, S_\mathbf{u})$ of the pseudo targets w.r.t. the inducing points Z is given by

$$\mathbf{m_u} = K_{ZZ}(K_{ZZ} + C)^{-1}\mathbf{c}, \qquad S_\mathbf{u} = K_{ZZ}(K_{ZZ} + C)^{-1}K_{ZZ} \tag{3.2.5}$$

which can be computed within $O(nm^2)$ time.

To reiterate, we propose to use subsampling for efficient mini-batch training while employing variational conditioning (or other scalable inference methods) during inference to

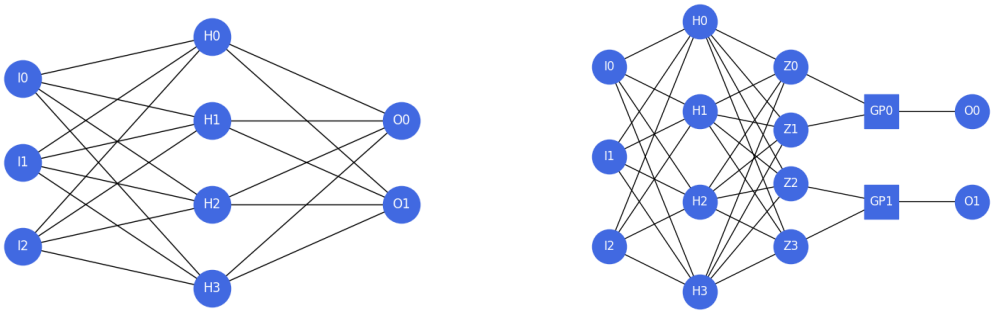
[2]It is important to note that the subsampling strategy for GPs assumes the batches are sampled randomly [35] or quasi-uniformly [98] from the dataset. In principle, we should not use the transitions sampled from prioritized experience replay (PER) [63] used by TD-MPC.

maintain constant-time predictions. This approach might seem counterintuitive, as the approximation schemes are different during the training and inference stages. However, it is motivated by the insight that GP training primarily involves learning hyperparameters—that is, establishing a function prior that best explains the data—whereas GP inference is inherently more data-dependent due to its non-parametric nature.

For inducing point selection, we could employ the pivoted Cholesky method [34], which selects up to m inducing points in $O(nm^2)$ time and may terminate early if the truncation error, controlled by the trace norm of the Schur complements in the Cholesky decomposition steps, falls below the specified tolerance. Moreover, when m is large, we may employ farthest point sampling (FPS) [25], which selects a low-discrepancy subsampled point set by balancing between filling up the space and separating the selected points. Notably, FPS requires $O(nm)$ time computation and $O(n)$ storage, which is more efficient than pivoted Cholesky decomposition. Therefore, when dealing with large datasets, we may first use FPS to select a low-discrepancy candidate set, then apply pivoted Cholesky upon them.

3.3 Deep Kernel Learning with Parameter Sharing

We would also like to investigate the effect of deep kernel learning (DKL, as described in Section 2.3) [92] in mitigating the potential curse of dimensionality and enhancing the expressive power of standard kernels, such as the RBF and Matérn kernels. Furthermore, in contrast to previous applications of GPs in MBRL where independent GPs are employed for multi-dimensional outputs (i.e., dynamics and reward predictions), we share the hidden layer of the MLP feature extractor used in DKL, which may enhance sample efficiency and introduce dependencies between different outputs similar to MLPs (see Figure 3).



(a) (b)

Figure 3: Examples with 3-dimensional input and 2-dimensional output. (a) MLP architecture; (b) DKL architecture (with 2-dimensional features for each independent GP).

4 Results

Our implementation[3] of the GP models for dynamics and reward prediction is built upon the GPyTorch library [28], allowing us to train a batch of GPs in parallel. However, we do not use the implementation of online variational conditioning (OVC) [47] available in GPyTorch due to integration challenges with the rest of our framework. Instead, we directly implement the variational conditioning strategy described by equations (3.2.4) and (3.2.5), leveraging GPyTorch's native support for numerical linear algebra, including the pivoted Cholesky method from its LinearOperator package.

We compare different variants of GP-TD-MPC with the TD-MPC (no latent) baseline across five OpenAI Gymnasium environments [78] (see Figure 4). All experiments were conducted on a ROG Zephyrus M16 (2023) GU604 laptop equipped with an Intel i9-13900H (2.60 GHz) CPU, 48GB of RAM, an NVIDIA RTX 4070 (8 GB) GPU, and an Intel Iris Xe graphics (23.8 GB).[4]

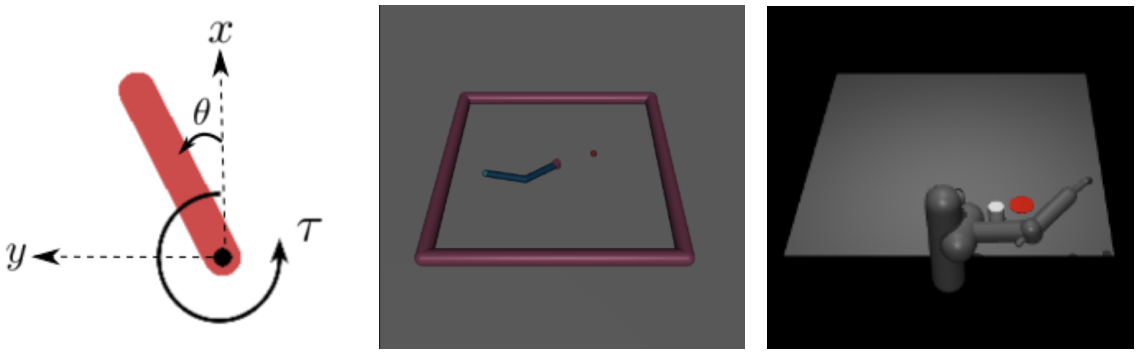
Unless otherwise stated, we use the hyperparameter settings similar to those specified in TD-MPC [33] with the following modifications:

- Number of Inducing Points: $m \leq 256$
- Pivoted Cholesky Error Tolerance: 1e$-$6
- Batch Size: 256 for both MLP and GP training
- DKL Features: 3-dimensional features for each independent GP
- Hidden Dimensions: 256 hidden units for all MLPs (including DKL components)
- Initial Rollouts: $2 \sim 8$ random rollouts
- Planning Horizon: $H = 5$
- Exploration Scheduler (ϵ): Linearly decayed from 0.5 to 0.05 over the first 2/3 of the total training steps

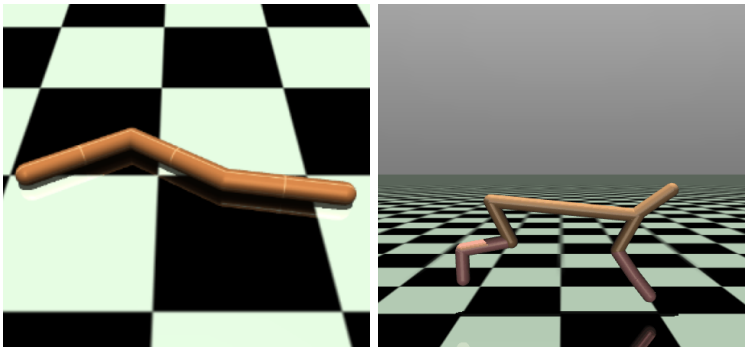
Each method (i.e., GP-TD-MPC with different kernels) was repeated five times across different random seeds to ensure robustness in the results, although some methods, particularly those using the SM kernel, were omitted in certain environments due to their prohibitive runtime and memory requirements. The learning curves for different methods we demonstrated in this section would represent the average of the highest total rewards observed up to the current timestep.

[3] The code and hyperparameter settings are available at `https://github.com/hanyang-hu/gp-mbrl`.

[4] For computations requiring extensive GPU memory, the NVIDIA GPU is prioritized; if its memory is exhausted, the integrated graphics' shared memory is used as a fallback.



(a) Pendulum (b) 2-DOF Reacher (c) 7-DOF Pusher



(d) Swimmer (e) Half Cheetah

Figure 4: Overview of the experiment environments in OpenAI Gymnasium [78].

Furthermore, to ensure reproducibility across trials with identical random seeds, we would like to limit the number of sources of nondeterministic behavior in the environment. Therefore, we configure PyTorch [54] to use deterministic algorithms, although they are often slower than their nondeterministic alternatives.

4.1 Inverted Pendulum Swingup

The inverted pendulum swingup problem (Figure 4(a)) is a classic continuous-control problem. The system involves a pendulum with one fixed end. Starting with a random initial pose, the objective is to apply torque to the other free end to swing the pendulum upward until it stabilizes in an upright position. The observation is a 3-dimensional tuple $(x, y, \dot{\theta})$, which is composed of the coordinates of the pendulum's free end and its angular velocity. The action should be a torque $\tau \in [-2, 2]$ applied to the pendulum. The ground-truth reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined by

$$r((x, y, \dot{\theta}), \tau) = -(\theta^2 + 0.1 \times \dot{\theta}^2 + 0.001 \times \tau^2) \tag{4.1.1}$$

where $\theta \in [-\pi, \pi]$ is the pendulum's angle obtained from the coordinates (x, y), with $\theta = 0$ meaning the pendulum is in the upright position.

We conducted experiments on the Pendulum-v1 environment over 15 episodes, beginning with 4 episodes of random rollouts. The results, illustrated in Figure 5, demonstrate that GP-TD-MPC with the RBF kernel and GP-TD-MPC with the SM kernel (combined with DKL) consistently outperform the TD-MPC baseline.

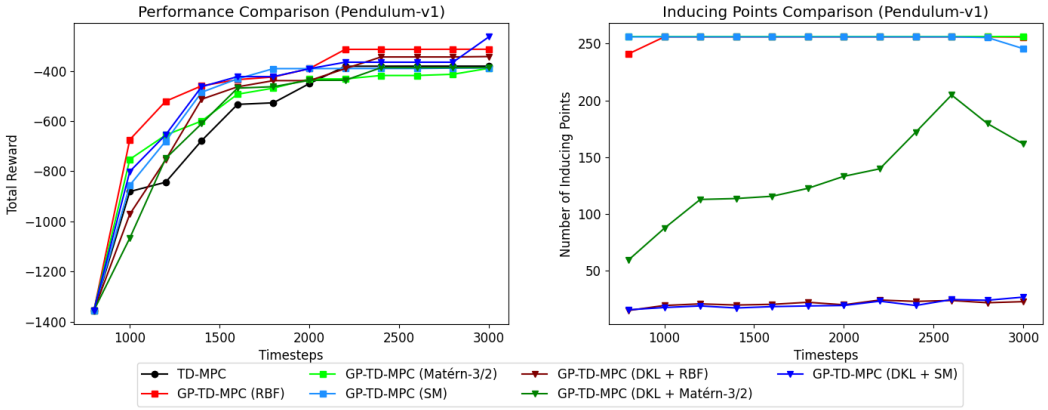


Figure 5: Comparison of the performance and number of inducing points (at most 256) in the Pendulum-v1 environment. GP-TD-MPC with the RBF kernel and the DKL+SM kernel performed better than the TD-MPC baseline. In the meantime, all variants of GP-TD-MPC outperform the baseline in the first 2000 timesteps.

4.2 Multi-Joint Dynamics with Contact (MuJoCo)

MuJoCo (Multi-Joint Dynamics with Contact) [77] is a general-purpose physics engine designed for efficient and accurate simulation of articulated structures interacting with their environments, particularly through physical contacts. We perform experiments on MuJoCo environments available in the OpenAI Gymnasium [78]. However, due to the deprecation of Windows support for MuJoCo's Python bindings, all experiments are conducted on the Windows Subsystem for Linux (WSL).

4.2.1 2-DOF Reacher

The "2-DOF Reacher", illustrated in Figure 4(b), is a 2-joint robot arm that attempts to move the robot's end effector (i.e., the fingertip) close to a target. The observation space is $\mathcal{S} = \mathbb{R}^{10}$, where each observation $s \in \mathcal{S}$ is composed of cosine/sine of the angles of the two arms, the angular velocities of the two arms, the coordinates of the target, and the vector between the target and the reacher's fingertip. The action space is $\mathcal{A} = [-1, 1]^2$, which represents the torques applied to the two hinge joints. The reward is defined by the negative of the distance between the fingertip and the target, plus an action penalty.

We conducted experiments on the Reacher-v5 environment over 20 episodes, starting with 2 episodes of random rollouts. The results, shown in Figure 6, demonstrate that all variants of GP-TD-MPC deliver comparable performance to the baseline.

4.2.2 7-DOF Pusher

The "7-DOF Pusher", illustrated in Figure 4(c), is a multi-jointed robot arm that is significantly more complex compared to the "2-DOF Reacher" aforementioned. Its objective is to push a target cylinder (referred to as the object) to a goal position using the robot's fingertip. The observation space is $\mathcal{S} = \mathbb{R}^{22}$ and the action space is $\mathcal{A} = [-2, 2]^7$, both of which have the highest dimensionality among the tasks we consider in this report. The reward incentivizes keeping the fingertips close to the object and positioning the object near the goal. Similar to the "2-DOF Reacher", it includes an action penalization term.

We conducted experiments on the Pusher-v5 environment over 30 episodes, beginning with 8 episodes of random rollouts. Variants of GP-TD-MPC using the SM kernel are omitted hereafter due to prohibitive runtime and memory demands. The results are summarized in Figure 7, with all variants of GP-TD-MPC performing similarly to the baseline.

4.2.3 Swimmer

Depicted in Figure 4(d), the "Swimmer" [17] is a robotic system composed of three segments connected by two rotors. Operating in a two-dimensional pool, its objective is to propel itself rapidly to the right by applying torques to the rotors and leveraging fluid friction. The observation space is $\mathcal{S} = \mathbb{R}^8$, where each state $s \in \mathcal{S}$ consists of the

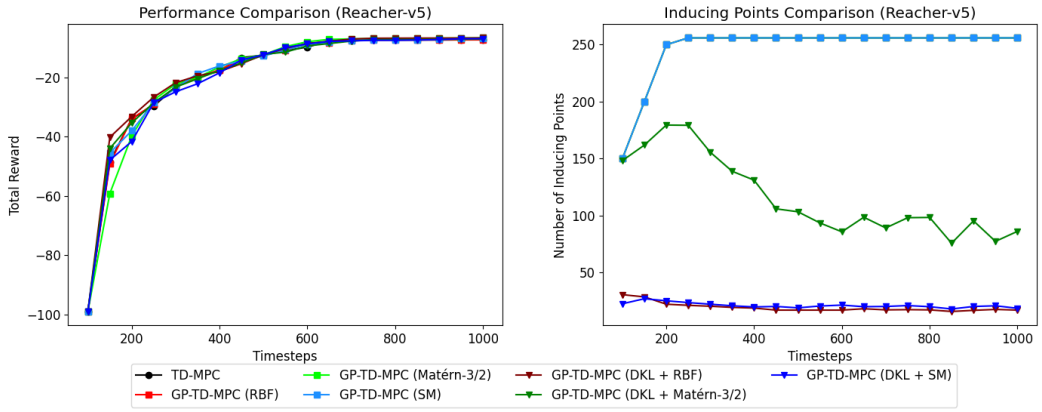


Figure 6: Comparison of the performance and number of inducing points (at most 256) in the Reacher-v5 environment.

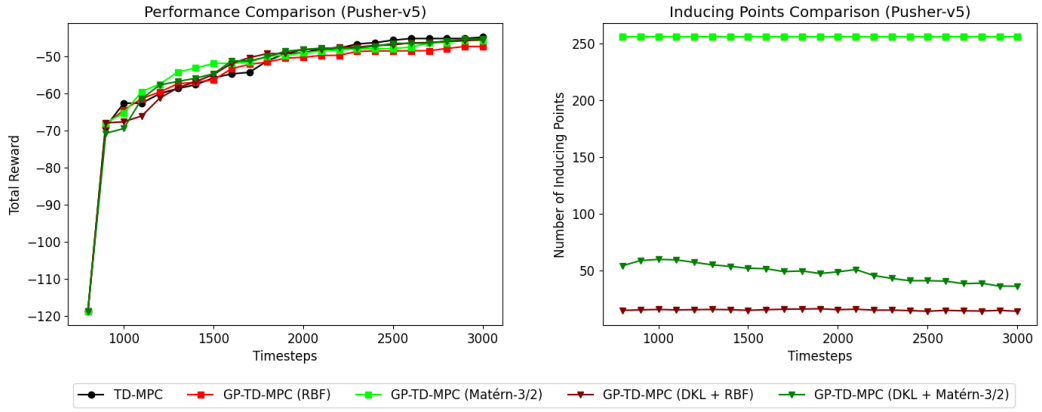


Figure 7: Comparison of the performance and number of inducing points (at most 256) in the Pusher-v5 environment.

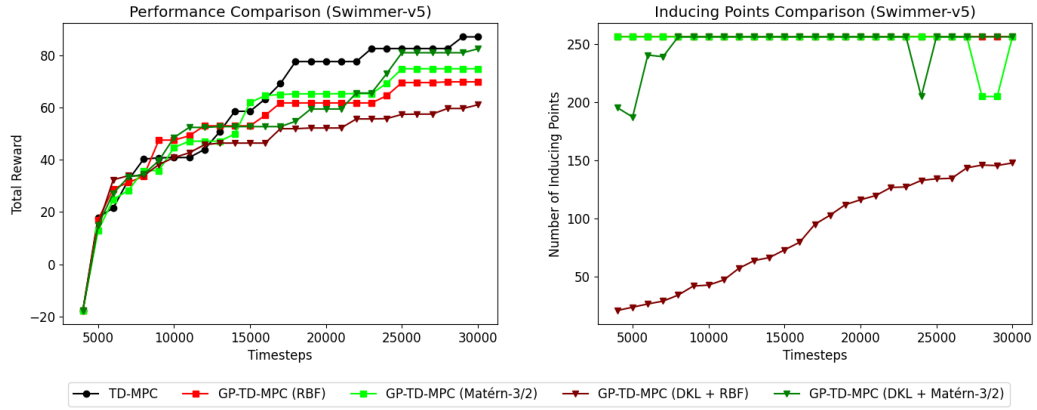


Figure 8: Comparison of performance and number of inducing points (up to 256) in the Swimmer-v5 environment. Numerical issues that occurred at the variational conditioning step for GP-TD-MPC with the Matérn kernels are reflected in the observed drop in the number of inducing points (in the right figure).

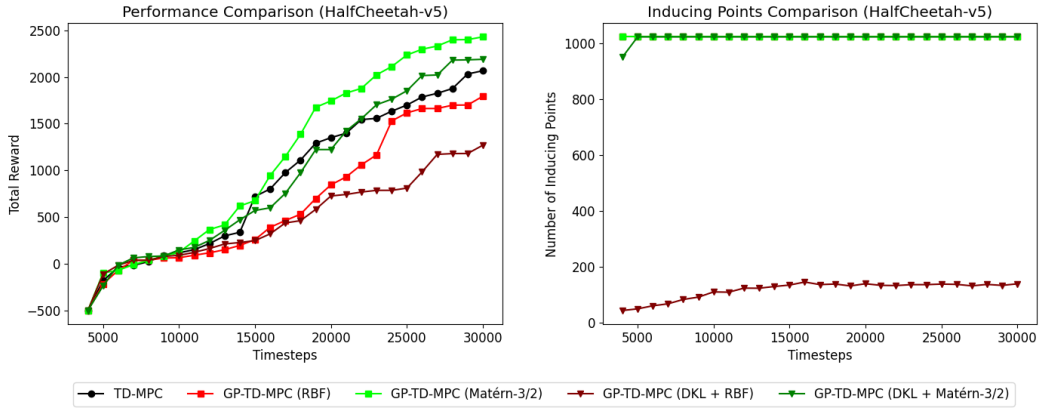


Figure 9: Comparison of performance and number of inducing points (up to 1024) in the HalfCheetah-v5 environment.

position values (i.e., angles) of the robot's body parts and the corresponding (angular) velocities. The absolute position of the swimmer in the two-dimensional pool is excluded to introduce the inductive bias that the optimal policy could be position-agnostic. The action space is $\mathcal{A} = [-1, 1]^2$, which represents the torques applied to the two rotors. The reward incentivizes the swimmer to move forward with action penalization.

We conducted experiments on the Swimmer-v5 environment over 30 episodes, beginning with 4 episodes of random rollouts. To reduce the total runtime, we implemented an action repeat of 2, where each action output by the agent is executed twice consecutively in the environment. Since the training dataset eventually grows to 1.5×10^4 samples, we employed FPS to select 4096 candidate points from the training data before applying the pivoted Cholesky decomposition to avoid memory overflow, as described in Section 3.2. The results are presented in Figure 8. All variants of GP-TD-MPC fail to outperform the baseline.[5]

4.2.4 Half Cheetah

Depicted in Figure 4(e), the "Half Cheetah" [87] is a 2-dimensional robot with 9 body parts and 8 joints. The observation space is $\mathbb{R}^{17}$ and the action space is $\mathcal{A} = [-1, 1]^6$ (torques are only applied to 6 joints excluding the head and the torso). The reward incentivizes the half cheetah to move forward as fast as possible with action penalization.

We conducted experiments on the HalfCheetah-v5 environment over 30 episodes, beginning with 4 episodes of random rollouts. Similar to the settings in the Swimmer-v5 environment, we implemented an action repeat of 2. However, we increase the limit of inducing points to $m \leq 1024$. The results are shown in Figure 9. GP-TD-MPC with the Matérn kernel consistently outperforms the baseline.

4.3 Key Findings and Insights

From the experimental results, we derive the following observations:

1. GP-TD-MPC with DKL typically requires fewer inducing points to achieve the error tolerance for the pivoted Cholesky method.

2. GP-TD-MPC with the Matérn-3/2 kernel consistently matches or even outperforms the TD-MPC baseline.

Our first observation suggests that DKL facilitates a more structured and data-adaptive feature representation, hence enabling the pivoted Cholesky method to identify a sparser subset of inducing points that could efficiently approximate the full GP posterior. Such efficiency gains align with the objectives of real-time planning. However, this comes at the cost of sample efficiency (particularly in the low-data regime) due to the additional

[5] We have also tried to increase the limit of inducing points to $m \leq 1024$, although we did not observe any performance improvements.

parameters required for training an MLP feature extractor. We have observed that GP-TD-MPC variants with DKL typically perform worse than their standard kernel counterparts.

For the second observation, we hypothesize that the Matérn kernel is better at capturing the sharp changes in the dynamics due to contacts, which should be prevalent in tasks like HalfCheetah-v5. To illustrate this, we visualize a toy example in Figure 10. In particular, we consider the following two-dimensional trajectory of a bouncing ball:

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} v_x t \\ v_y \left(t - T\lfloor \frac{t}{T} \rfloor\right) - \frac{1}{2} g \left(t - T\lfloor \frac{t}{T} \rfloor\right)^2 \end{pmatrix} \tag{4.3.1}$$

where v_x, v_y are constants and $T = 2v_y/g$. We have trained GP models with the RBF kernel and the Matérn-3/2 kernel[6], as well as an MLP model with 256 hidden units and the ELU$(\cdot)$ activation function (as in the settings of TD-MPC [33])

$$\mathrm{ELU}(x) = \begin{cases} x, & \text{if } x > 0, \\ \alpha\big(e^x - 1\big), & \text{if } x \leq 0. \end{cases} \tag{4.3.2}$$

which is only once differentiable everywhere with $\alpha = 1$ [16]. The dataset consists of 500 points on the trajectory. The GP models are trained with 200 iterations and the MLP model is trained with 2×10^4 iterations. We may observe that the GP model with the Matérn kernel provides the best prediction result. On the other hand, the variant with the RBF kernel fails to produce reliable predictions in regions lacking training data.

One possible explanation for this phenomenon is that the RBF kernel is smooth, imposing assumptions that are unrealistic for many physical processes [61]. However, we could also understand this phenomenon from a frequency-domain perspective: the spectral densities of the RBF kernel and the Matérn kernel are given by

$$\begin{aligned} p_l^{\mathrm{RBF}}(s) &= (2\pi l^2)^{D/2} \exp\left(-2\pi^2 l^2 s^2\right) \\ p_{l,\nu}^{\text{Matérn}}(s) &= \frac{2^D \pi^{D/2} \Gamma(\nu + D/2)(2\nu)^\nu}{\Gamma(\nu) l^{2\nu}} \left(\frac{2\nu}{l^2} + 4\pi^2 s^2\right)^{-(\nu + D/2)} \end{aligned} \tag{4.3.3}$$

which resembles the probability density functions of multivariate Gaussian distribution and multivariate t-distribution (with $2\nu + D$ degrees of freedom) respectively [61].[7] Since the latter has heavier tails, the Matérn kernel is more robust in accommodating high-frequency "outliers" compared to the RBF kernel. As shown in Figure 10, to capture the abrupt changes in direction caused by contact events, the RBF kernel compensates by overfitting to an excessively small lengthscale, resulting in poor generalization.

[6] We have also tried out the SM kernel, although it did not work in this case.

[7] We consider a 1-D lengthscale hyperparameter $l \in \mathbb{R}_{>0}$ without automatic relevance determination [88].

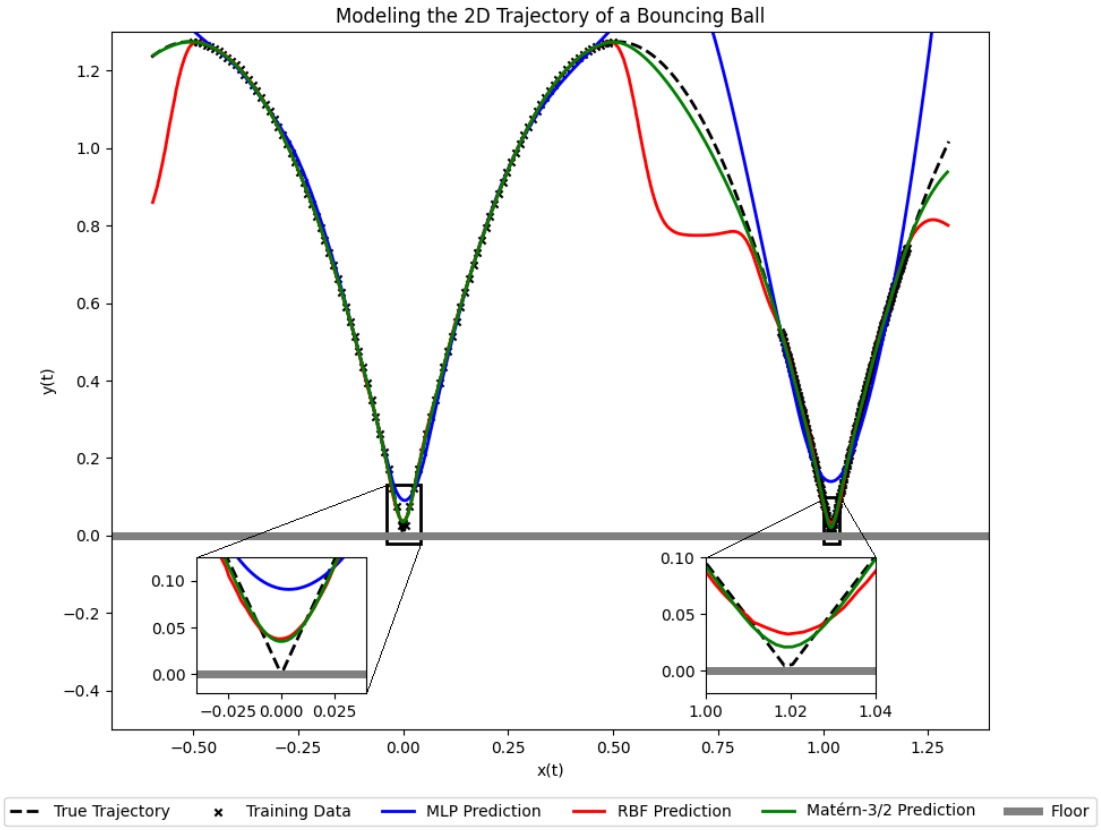


Figure 10: Visualization of the Matérn kernel's ability to model contact dynamics.

5 Discussion

5.1 Runtime and Efficiency

Table 1 presents the total runtime of TD-MPC and various GP-TD-MPC variants across different tasks, averaged over five trials. For lower-dimensional tasks such as Pendulum-v1, Reacher-v5, and Swimmer-v5, the runtime of GP-TD-MPC generally remains within 1.5 times that of the baseline.[8] Notably, in the Swimmer-v5 task, increasing the maximum number of inducing points to $m \leq 1024$ does not result in a significant runtime increase, suggesting that the GPU's parallel computation capabilities have yet to be fully utilized. However, for tasks with higher dimensionality and larger training datasets, such as HalfCheetah-v5 (where $\mathcal{S} = \mathbb{R}^{17}$ and takes 1.5×10^5 data points), GP-TD-MPC typically requires 2–3 times the runtime of the baseline, except for GP-TD-MPC (RBF+DKL).

We have identified several factors in our implementation that may hinder the computational efficiency of GP-TD-MPC, both in comparison to the baseline and in potential real-world applications:

- The SVGP correction in equation (3.2.3) could be computed in parallel (e.g., via multiprocessing) with the inference of the base model f_θ, further reducing the runtime. However, for simplicity, our implementation executes the SVGP correction sequentially after the inference of the base model.
- To facilitate reproducibility, we configure PyTorch to use deterministic algorithms, which are generally slower than their nondeterministic counterparts.
- Running simulations on Windows Subsystem for Linux (WSL) may be less efficient compared to native Linux environments due to the additional virtualization overhead.

5.2 Alternative Inference Methods

It has been shown by Titsias [76] that selecting more data points as inducing points would never decrease the ELBO (3.2.1). Intuitively, larger training datasets may benefit more from a higher limit of inducing points. However, further increasing the number of inducing points can result in prohibitive runtime and memory requirements, particularly for real-time applications on mobile computing platforms. In this subsection, we explore alternative inference methods that could (1) either scale more effectively with a larger number of inducing points; or (2) make more efficient use of the limited inducing points.

[8] Excluding the SM kernel, which requires noticeably longer runtime and memory, trading increased computational cost for potentially greater expressiveness—though we did not observe any practical benefit.

5.2.1 Local Kernel Interpolation

We consider the local kernel interpolation (LKI) method [93] which has $O(1)$-time cached mean prediction, insensitive to the number of inducing points.

Specifically, assuming that the m inducing points Z form a grid, then the Nyström approximation [89] could be further approximated by

$$K \approx K_{XZ}K_{ZZ}^{-1}K_{ZX} \approx W^\top K_{ZZ}W \tag{5.2.1}$$

where $W \in \mathbb{R}^{m\times n}$ is a sparse interpolation matrix such that $K_{ZZ}W \approx K_{ZX}$. Therefore, we could approximate the GP correction (3.1.4) by

$$\text{output}(\mathbf{x}^*) = f_\theta(\mathbf{x}^*) + \underbrace{\mathbf{w}_{\mathbf{x}^*}^\top K_{ZZ}W(W^\top K_{ZZ}W + \sigma_\epsilon^2\mathbf{I})^{-1}(\mathbf{y} - f_\theta(X))}_{\text{LKI Correction}} \tag{5.2.2}$$

where $\mathbf{w}_{\mathbf{x}^*} \in \mathbb{R}^m$ is a sparse interpolation vector such that $K_{ZZ}\mathbf{w}_{\mathbf{x}^*} \approx \mathbf{k}_{Z\mathbf{x}^*}$. The term $K_{ZZ}W(W^\top K_{ZZ}W + \sigma_\epsilon^2\mathbf{I})^{-1}(\mathbf{y} - f_\theta(X))$ can be cached, allowing the LKI-based correction to be computed in $O(1)$-time (since $\mathbf{w}_{\mathbf{x}^*}$ is sparse), unlike the SVGP correction in equation (3.2.3) which requires $O(m)$-time computation.

Despite the computational benefits with a larger number of inducing points, the LKI approach comes with additional overhead during the interpolation step (to compute $\mathbf{w}_{\mathbf{x}^*}$), which could be observed from the empirical comparisons in Figure 2 and Figure 11. Furthermore, the grid structure of inducing points is prone to the curse of dimensionality, since the number of inducing points required to densely cover the input space increases exponentially with the input's dimensionality. In high-dimensional settings, this exponential growth can substantially raise memory requirements. Therefore, DKL [92] or additive kernels [23] are usually employed when using LKI.

Moreover, the LKI approach requires all data to be contained in a pre-specified grid where the inducing points are distributed. In the GPyTorch [28] implementation, the data is scaled to fit within the grid's bounds using the following method:

$$\text{ScaleToBound}(\mathbf{x}) = \sigma\,(\ell_u - \ell_l)\cdot \underbrace{\frac{\mathbf{x} - \mathbf{x}_{\min}}{\mathbf{x}_{\max} - \mathbf{x}_{\min}}}_{\text{element-wise division}} + \sigma\,\ell_l \tag{5.2.3}$$

where $\ell_b, \ell_u \in \mathbb{R}$ are the lower and upper bounds of the grid $[\ell_b, \ell_u]^D$, $\sigma \in (0, 1)$ is a scaling factor to ensure numerical stability for interpolation, and $\mathbf{x}_{\min}, \mathbf{x}_{\max} \in \mathbb{R}^D$ are vectors where each element represents the smallest (resp. largest) observed value of the input data in the corresponding dimension.

When training in mini-batch, the kernel approximation in equation (5.2.1) may be unnecessary. Instead, we can adopt the hybrid strategy described in Section 3.2, where we optimize the hyperparameters for an exact GP over subsampled data during training, then employ the approximated correction (5.2.2) with the optimized hyperparameters during

inference. However, if we are using the ScaleToBound$(\cdot)$ method in (5.2.3) with DKL, then we need to recompute $\mathbf{x}_{\min}$ and $\mathbf{x}_{\max}$ after each gradient update, which would incur an $O(n)$ computational cost.[9] To facilitate the use of subsampling during training, we would like to have a ScaleToBound$(\cdot)$ method that is insensitive to changes in data embeddings. Consequently, we consider the following data-independent ScaleToBound$(\cdot)$ method

$$\text{ScaleToBound}(\mathbf{x}) = \sigma \tanh(\mathbf{x}) \tag{5.2.4}$$

which maps all data into a fixed grid $[-\sigma, \sigma]^D$. When combined with DKL, this can be regarded as an activation in the final layer of the MLP feature extractor, ensuring that the transformed features remain within the fixed grid.

We evaluated our implementation of the LKI-based correction (5.2.2) on the Swimmer task. In this setting, all variants of GP-TD-MPC employing the variational conditioning strategy (described in Section 3.2) failed to outperform the baseline. To address the curse of dimensionality, we employed DKL together with the ScaleToBound$(\cdot)$ method in (5.2.4) to map all input data into fixed 2-dimensional grids for each independent GP, and we used $m = 150^2 = 2.25 \times 10^4$ inducing points, which exceed the total number of data points. Only the variant using the SM kernel outperformed the baseline during the first half of the experiment and delivered comparable performance overall. However, unlike previous observations, using the SM kernel with LKI did not incur substantial memory or runtime requirements. This is because the sparse interpolation allows LKI-based correction to only access $O(1)$ entries of the cached vector. The results are presented in Figure 11.

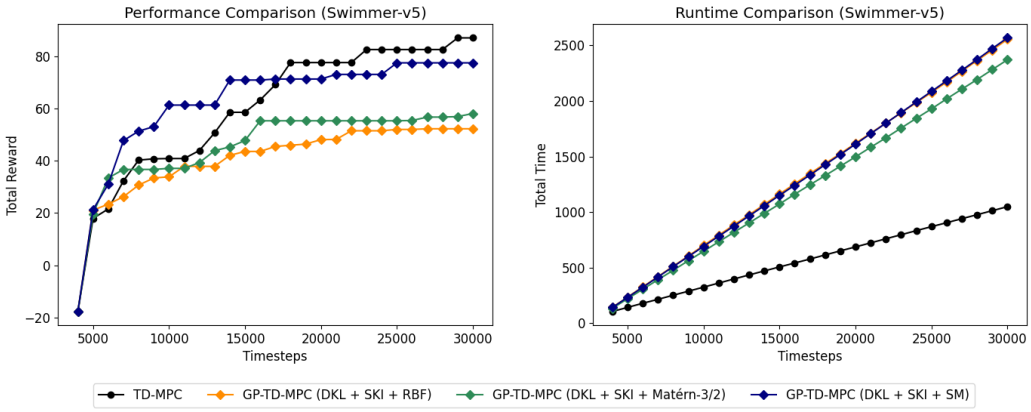


Figure 11: Comparison of performance and total runtime in the Swimmer-v5 environment.

Remark. We have observed that our implementation of the LKI-based correction is not deterministic, leading to variations in results across different trials, even with the same

[9] Notice that the latent embeddings of the dataset would change after each update.

random seed. We hypothesize that this is due to GPyTorch's implementation of SKI relying on operations that lack deterministic alternatives.

5.2.2 Dynamical Local Projection

Based on the intuition that MPC locally optimizes the trajectory, inducing points closer to the initial trajectory are likely to be more informative. For example, Hewing et al. [37] propose dynamically selecting inducing points at each sampling time based on their proximity to the optimized trajectory from the previous timestep. However, they employ a low-rank kernel approximation strategy called the Fully Independent Training Condition (FITC) [58], which may be considered less favorable compared to SVGPs [5]. Consequently, we would like to similarly select inducing points for variational conditioning.

Instead of re-computing the variational conditioning in $O(nm^2)$ time at each timestep, We consider utilizing pre-computed $\mathbf{c}$ and C from equation (3.2.4) with a large set of inducing points Z of size M, then project them to a subset of inducing points $Z' \subseteq Z$ of size m that are closer to the reference trajectory $\mathbf{p}$ in $O(M^2 m)$ time, which is independent of the size of the dataset. We will refer to this method as dynamical local projection (DLP).

DLP is inspired by online variational conditioning (OVC) [47]. Specifically, we have

$$\begin{aligned} \mathbf{c}' &= K_{Z'X}\Sigma_{\mathbf{y}}^{-1}\mathbf{y} \approx K_{Z'Z}(K_{ZZ}^{-1}\mathbf{c}) \\ C' &= K_{Z'X}\Sigma_{\mathbf{y}}^{-1}K_{XZ'} \approx K_{Z'Z}(K_{ZZ}^{-1}CK_{ZZ}^{-1})K_{ZZ'} \end{aligned} \tag{5.2.5}$$

where $K_{ZZ}^{-1}\mathbf{c} \in \mathbb{R}^M$ and $K_{ZZ}^{-1}CK_{ZZ}^{-1} \in \mathbb{R}^{M\times M}$ are cached. We may subsequently use $\mathbf{c}'$ and C' to compute the optimal variational distribution by equation (3.2.5).

The parameter projection (5.2.5) could be interpreted by the Nyström approximation (see Appendix B.2.2). In particular, we would like to find an efficient approximation (i.e., without requiring the access to the full dataset) of the projection matrix $P = K_{Z'X}(K_{XZ}K_{ZX})^{+}K_{XZ}$ such that $\mathbf{c}' = P\mathbf{c}$ and $C' = PCP^{\top}$.[10] If we use the Nyström approximation $K_{Z'X} \approx K_{Z'Z}K_{ZZ}^{-1}K_{ZX}$, then it is easy to see that

$$\begin{aligned} P &= K_{Z'X}(K_{XZ}K_{ZX})^{+}K_{XZ} \\ &\approx K_{Z'Z}K_{ZZ}^{-1}K_{ZX}\cancel{(K_{XZ}K_{ZX})^{+}K_{XZ}} \\ &= K_{Z'Z}K_{ZZ}^{-1} \end{aligned} \tag{5.2.6}$$

by properties of the pseudo inverse of products, assuming that K_{ZX} has full row rank.

For the selection of $Z' \subseteq Z$, we consider the top-m candidates of Z based on the following heuristic cost function (lower is better)

$$\text{cost}(\mathbf{z}, \mathbf{p}) = \min_{h=1,\dots,H-1} \eta^h \text{dist}(\mathbf{z}, \overline{\mathbf{p}_h\mathbf{p}_{h+1}}) \tag{5.2.7}$$

[10] We consider the pseudo inverse $(K_{XZ}K_{ZX})^{+}$ since $K_{XZ}K_{ZX}$ is a low-rank matrix.

where $\mathbf{p}$ is the reference path (which we choose to be the optimized trajectory of the previous timestep). The term $\overline{\mathbf{p}_h\mathbf{p}_{h+1}}$ represents the line segment connecting the h-th and $(h+1)$-th points in the reference path $\mathbf{p}$, and $\eta \in (0,1)$ is a temporal discount factor which encourages more points to be selected near line segments corresponding to larger timesteps. A visualized example is presented in Figure 12.

We conducted an empirical comparison of three inference methods—standard variational conditioning, LKI, and DLP—on the Pendulum-v1 environment. The results, shown in Figure 13, were obtained with hyperparameters $m = 128$, $M = 2048$, and $\eta = 0.8$. While DLP achieved faster convergence than the TD-MPC baseline similar to other variants of GP-TD-MPC, it performed worst among all other variants in this experiment. Notably, however, DLP incurred significantly lower computational overhead compared to LKI.

5.3 Potential Extensions

In this subsection, we discuss several approaches to further enhance the efficiency of GP-TD-MPC, which we leave the exploration to future work.

5.3.1 Inducing Point Allocation

The inducing point allocation (IPA) methods that we use, FPS [25] (kernel-independent) and pivoted Cholesky decomposition [34] (kernel-dependent yet target-independent), share a common limitation: neither fully exploits the joint information in the dataset $(X, \mathbf{y})$ to optimize inducing point allocation. This is particularly suboptimal in our setting, where the GP model serves as a correction to the MLP base model. Ideally, inducing points should be allocated (whilst maintaining diversity) to regions where predictions of the MLP base model f_θ deviate significantly from the ground-truth outputs, as these regions likely require stronger GP correction.

One possible approach is to sample from a determinantal point process (DPP, see Appendix B.4) over the dataset $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$ similar to a recent work on IPA for Bayesian optimization (BO) [50]. For example, consider an "importance" function related to the MLP model residuals

$$g(\mathbf{x}, y) = \alpha(\|f_\theta(\mathbf{x}) - y\|) \tag{5.3.1}$$

where $\alpha : \mathbb{R}_{\geq 0} \to \mathbb{R}_{>0}$ is monotonically increasing (e.g., $\alpha(x) = x + \epsilon$ for $\epsilon > 0$). Notice that the importance function $g(\cdot, \cdot)$ assigns higher values to data pairs for which the MLP base model produces inaccurate predictions. Correspondingly, we may define the following augmented kernel function

$$k_{\text{DPP}}((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2)) = g(\mathbf{x}_1, y_1)k(\mathbf{x}_1, \mathbf{x}_2)g(\mathbf{x}_2, y_2) \tag{5.3.2}$$

which is clearly symmetric and positive semi-definite. Intuitively, the sample from the DDP w.r.t. the augmented kernel function k_{DPP} defined in (5.3.2) should balance between diversity (captured by the GP's kernel k) and the importance (captured by g).

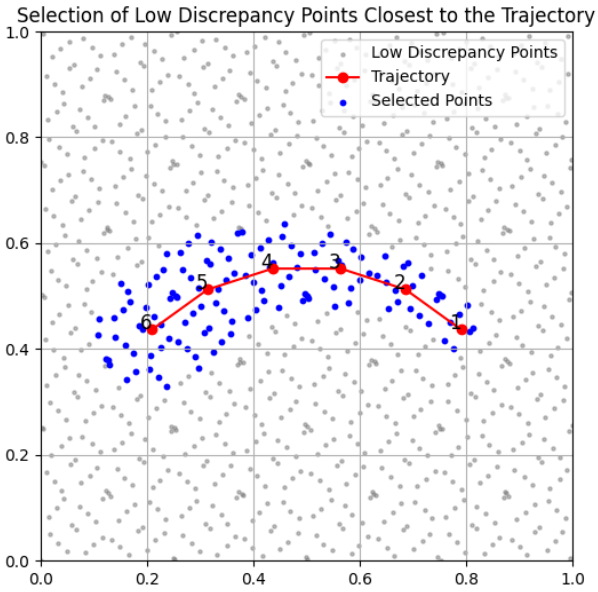


Figure 12: Selection of 128 points from 1024 Sobol points that are closest to the reference trajectory according to the heuristic in (5.2.7). A temporal discount factor of $\eta = 0.8$ encourages more points to be selected near line segments corresponding to larger timesteps, where the trajectory is expected to exhibit greater variation during the MPC optimization.

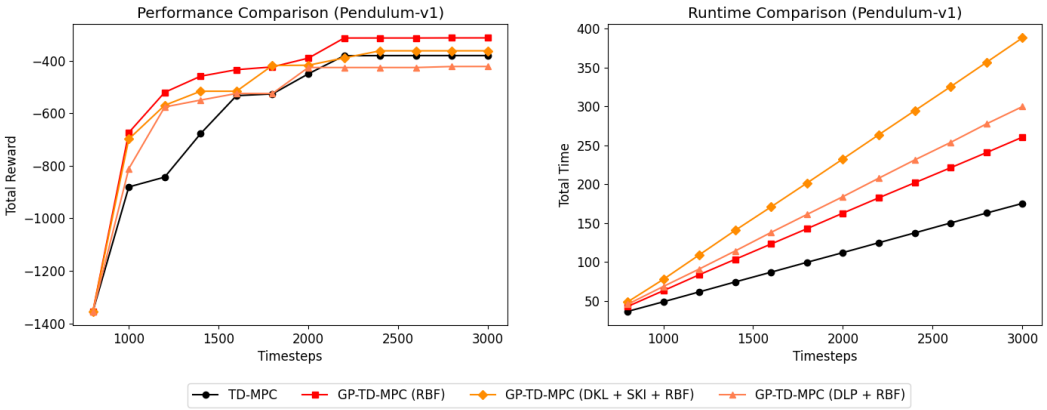


Figure 13: Comparison of performance and total runtime in the Pendulum-v1 environment.

5.3.2 Uncertainty Quantification

The ability of uncertainty quantification has been shown to mitigate the issue of the dynamics bottleneck in MBRL [15, 39, 83, 84]. However, Theorem 1 indicates that the dynamics gap for a short planning horizon ($H = 5$) is substantially smaller than for longer horizons ($20 \leq H \leq 40$, as in [83]). This reduced gap suggests that a deterministic model may suffice in practice, aligning with the empirical success of TD-MPC [33], which achieves strong performance without explicit stochastic modeling. Furthermore, we should note that the IPA strategy we mentioned in Section 5.3.1 may not be easily compatible with uncertainty quantification: When the inducing points are concentrated in regions where the base model exhibits poor performance, the resulting sparse coverage of the inducing points in other regions could undesirably inflate predictive variances, even if those areas are where the model's predictions are most reliable. Nevertheless, the ability of uncertainty quantification may still be an advantage for guiding exploration [19, 70] or learning in the low-data regime [20], which could be offline or require fewer inducing points.

It is important to note that both the SVGP and LKI approaches used in this report allow for constant-time posterior prediction and/or sampling [55, 94, 95]. For example, pathwise conditioning [94, 95] allows efficient sampling of functions from the posterior by combining scalable GP inference methods such as SVGP with Fourier features (see Appendix B.2). Specifically, pathwise conditioning decomposes the GP posterior as a weight-space prior and a function-space update (or "correction") as follows

$$(f \mid \mathbf{u})(\mathbf{x}^*) \approx \underbrace{\phi(\mathbf{x}^*)^\top \mathbf{w}}_{\text{weight-space prior}} + \underbrace{\mathbf{k}_{Z\mathbf{x}^*}^\top K_{ZZ}^{-1}(\mathbf{u} - \Phi\mathbf{w})}_{\text{function-space update}} \tag{5.3.3}$$

where $\phi(\cdot) : \mathbb{R}^D \to \mathbb{R}^s$ is the feature map associated with the kernel (potentially, approximated by Fourier features [59]) with $s \ll n$ being the feature dimension, $\mathbf{w} \sim \mathcal{N}(0, \mathbf{I}_s)$ is sampled from the prior weight distribution, $\mathbf{u} \sim \mathcal{N}(\mathbf{m}_\mathbf{u}, S_\mathbf{u})$ is sampled from the variational distribution in $O(m^2)$ time, and $\Phi = \phi(Z) \in \mathbb{R}^{m \times s}$ is the feature map evaluated at the m inducing points Z. Therefore, the uncertainty of pathwise conditioning for SVGP is mainly introduced by the weight-space prior $\mathcal{N}(0, \mathbf{I}_s)$, and controlled by the variational distribution $\mathcal{N}(\mathbf{m}_\mathbf{u}, S_\mathbf{u})$. Each evaluation of the sampled function takes $O(s + m)$ time after pre-computation.

We can easily integrate pathwise conditioning within our GP-TD-MPC framework as the inference is also based on SVGP correction (3.2.3)

$$(f \mid \mathbf{u})(\mathbf{x}^*) \approx \underbrace{f_\theta(\mathbf{x}^*) + \phi(\mathbf{x}^*)^\top \mathbf{w}}_{\text{function prior}} + \underbrace{\mathbf{k}_{Z\mathbf{x}^*}^\top K_{ZZ}^{-1}(\mathbf{u} - \Phi\mathbf{w})}_{\text{correction term}}. \tag{5.3.4}$$

This comes with several advantages for model-based planning compared to other scalable GP posterior sampling methods (e.g., LOVE [55]): (1) the function draw is differentiable, which admits gradient-based method (e.g., [36]) other than CEM and MPPI; (2) combining

multiple function draws could be interpreted as an ensemble of models, which admits a more flexible choice of trajectory sampling schemes (e.g., **TS-**1 and **TS-**∞ in PE-TS [15]).

5.3.3 Kernel Composition

As discussed in Section 4.3, the Matérn kernel may be more suitable for modeling contact dynamics. In general, we argue that the choice of the kernel could significantly affect the performance of GP-TD-MPC. Consequently, we may want to explore a wider class of kernels and analyze how their covariance structure can be tailored to benefit specific applications of GP-TD-MPC. By composing kernels through addition or multiplication, we may encode prior knowledge about structural properties (e.g., linearity and periodicity) into the model, enhancing its predictive capability [24].

However, the pathwise conditioning method [94, 95] for uncertainty-aware modeling relies on an (approximate) finite-dimensional weight-space representation. Typically, this involves Fourier feature approximations [59], which are limited to stationary kernels. This undesirably restricts the use of composite kernels such as those involving linear kernels. Fortunately, we can integrate the Fourier feature approximation of stationary kernels with other non-stationary kernels that have finite-dimensional feature maps (e.g., the linear kernel) into a unified representation. For instance, given two kernels $k_1(\cdot,\cdot)$ and $k_2(\cdot,\cdot)$ with finite-dimensional feature maps $\phi_1(\cdot)$ and $\phi_2(\cdot)$, the corresponding feature maps of the additive and multiplicative kernels could be attained by concatenation ($\phi_{\text{add}}(x) = [\phi_1(x),\, \phi_2(x)]$) or tensor product ($\phi_{\text{mult}}(x) = \phi_1(x) \otimes \phi_2(x)$) respectively.

6 Conclusion

This report demonstrates that scalable GPs could be effectively integrated with model-based online planning frameworks such as TD-MPC to further enhance sample efficiency while maintaining real-time tractability in environments of moderate dimensionality. By employing GPs as corrective models to MLP base models, our method could attain performance comparable to or exceeding the TD-MPC baseline in the majority of tasks. Moreover, we have observed the superiority of Matérn kernels in modeling contact dynamics and the computational advantages of DKL (e.g., in the HalfCheetah-v5 environment).

Directions for future works may include adaptive inducing point allocation (e.g., dynamical local projection and DDP-based methods), uncertainty-aware modeling (e.g., via pathwise conditioning), and kernel selection tailored to domain-specific applications. These potential advancements would position GP-based MBRL as a versatile and efficient paradigm for real-time planning in diverse environments.

Task Name	TD-MPC	RBF		Matérn-3/2		Spectral Mixture	
		Standard	DKL	Standard	DKL	Standard	DKL
Pendulum	175.13	260.31	266.87	279.64	297.81	430.92	287.64
Reacher	127.96	172.53	173.18	173.31	174.48	856.31	182.29
Pusher	201.40	305.83	293.66	315.20	302.04	–	–
Swimmer ($m \leq 256$)	1049.06	1531.21	1603.92	1537.10	1635.02	–	–
Swimmer ($m \leq 1024$)	1049.06	1592.99	1603.02	1666.68	1732.88	–	–
Half Cheetah	1064.46	2019.56	1631.59	2794.53	2818.31	–	–

Table 1: Comparison of total runtime (in seconds) averaged across 5 trials.

References

[1] G Journel Andre and J Huijbregts C. *Mining Geostatistics*. Blackburn Press, West Caldwell, NJ, February 2004.

[2] Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning, 2021.

[3] Kavosh Asadi, Dipendra Misra, and Michael L. Littman. Lipschitz continuity in model-based reinforcement learning, 2018.

[4] Haim Avron, Vikas Sindhwani, Jiyan Yang, and Michael W. Mahoney. Quasi-monte carlo feature maps for shift-invariant kernels. *Journal of Machine Learning Research*, 17(120):1–38, 2016.

[5] Matthias Bauer, Mark van der Wilk, and Carl Edward Rasmussen. Understanding probabilistic sparse gaussian process approximations, 2017.

[6] Felix Berkenkamp, Angela P. Schoellig, and Andreas Krause. No-regret bayesian optimization with unknown hyperparameters. *Journal of Machine Learning Research*, 20(50):1–24, 2019.

[7] Alexei Borodin. Determinantal point processes, 2009.

[8] Nathanael Bosch, Jan Achterhold, Laura Leal-Taixé, and Jörg Stückler. Planning from images with deep latent gaussian process dynamics, 2020.

[9] Zdravko I. Botev, Dirk P. Kroese, Reuven Y. Rubinstein, and Pierre L'Ecuyer. *The Cross-Entropy Method for Optimization*, page 35–59. Elsevier, 2013.

[10] David R. Burt, Carl Edward Rasmussen, and Mark van der Wilk. Convergence of sparse variational inference in gaussian processes regression. *Journal of Machine Learning Research*, 21(131):1–63, 2020.

[11] Difeng Cai, James Nagy, and Yuanzhe Xi. Fast deterministic approximation of symmetric indefinite kernel matrices with high dimensional datasets, 2021.

[12] Konstantinos Chatzilygeroudis and Jean-Baptiste Mouret. Using parameterized black-box priors to scale up model-based policy search for robotics, 2018.

[13] Konstantinos Chatzilygeroudis, Roberto Rama, Rituraj Kaushik, Dorian Goepp, Vassilis Vassiliades, and Jean-Baptiste Mouret. Black-box data-efficient policy search for robotics, 2017.

[14] Laming Chen, Guoxin Zhang, and Hanning Zhou. Fast greedy map inference for determinantal point process to improve recommendation diversity, 2018.

[15] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models, 2018.

[16] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016.

[17] Rémi Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. Theses, Institut National Polytechnique de Grenoble - INPG, June 2002.

[18] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, October 2001.

[19] Sebastian Curi, Felix Berkenkamp, and Andreas Krause. Efficient model-based reinforcement learning through optimistic policy search and planning, 2020.

[20] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423, February 2015.

[21] Marc Peter Deisenroth and Carl Edward Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 465–472. Omnipress, 2011.

[22] Luc Devroye, Abbas Mehrabian, and Tommy Reddad. The total variation distance between high-dimensional gaussians with the same mean, 2023.

[23] Nicolas Durrande, David Ginsbourger, and Olivier Roustant. Additive kernels for gaussian process modeling, 2011.

[24] David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search, 2013.

[25] Y. Eldar, M. Lindenbaum, M. Porat, and Y.Y. Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6(9):1305–1315, 1997.

[26] Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with gaussian processes. In Luc De Raedt and Stefan Wrobel, editors, *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pages 201–208. ACM, 2005.

[27] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

[28] Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration, 2021.

[29] Jacob R. Gardner, Geoff Pleiss, Ruihan Wu, Kilian Q. Weinberger, and Andrew Gordon Wilson. Product kernel interpolation for scalable gaussian processes, 2018.

[30] Gene H Golub and Charles F Van Loan. *Matrix Computations*. Johns Hopkins Series in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 3 edition, October 1996.

[31] David Ha and Jürgen Schmidhuber. World models. 2018.

[32] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination, 2020.

[33] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control, 2022.

[34] Helmut Harbrecht, Michael Peters, and Reinhold Schneider. On the low-rank approximation by the pivoted cholesky decomposition. *Applied Numerical Mathematics*, 62(4):428–440, 2012. Third Chilean Workshop on Numerical Analysis of Partial Differential Equations (WONAPDE 2010).

[35] Kohei Hayashi, Masaaki Imaizumi, and Yuichi Yoshida. On random subsampling of gaussian process regression: A graphon-based analysis, 2019.

[36] Mikael Henaff, William F. Whitney, and Yann LeCun. Model-based planning with discrete and continuous actions, 2018.

[37] Lukas Hewing, Juraj Kabzan, and Melanie N. Zeilinger. Cautious model predictive control using gaussian process regression. *IEEE Transactions on Control Systems Technology*, 28(6):2736–2743, November 2020.

[38] J. Ben Hough, Manjunath Krishnapur, Yuval Peres, and Bálint Virág. Determinantal processes and independence. *Probability Surveys*, 3(none), January 2006.

[39] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization, 2021.

[40] R. Keys. Cubic convolution interpolation for digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(6):1153–1160, 1981.

[41] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[42] Alex Kulesza and Ben Taskar. k-dpps: fixed-size determinantal point processes. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 1193–1200, Madison, WI, USA, 2011. Omnipress.

[43] Mario Köppen. The curse of dimensionality. *5th online world conference on soft computing in industrial applications (WSC5)*, 2000.

[44] Armin Lederer, Jonas Umlauft, and Sandra Hirche. Uniform error bounds for gaussian process regression with application to safe control, 2019.

[45] Kevin Li, Max Balakirsky, and Simon Mak. Trigonometric quadrature fourier features for scalable gaussian process regression, 2023.

[46] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[47] Wesley J. Maddox, Samuel Stanton, and Andrew Gordon Wilson. Conditioning sparse variational gaussian processes for online decision-making, 2021.

[48] John Martin, Jinkun Wang, and Brendan Englot. Sparse gaussian process temporal difference learning for marine robot navigation. In Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 179–189. PMLR, 29–31 Oct 2018.

[49] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[50] Henry B. Moss, Sebastian W. Ober, and Victor Picheny. Inducing point allocation for sparse gaussian processes in high-throughput bayesian optimisation, 2023.

[51] Cameron Musco and Christopher Musco. Recursive sampling for the nyström method, 2017.

[52] Mojmir Mutny and Andreas Krause. Efficient high dimensional bayesian optimization with additivity and quadrature fourier features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[53] Gabriel Parra and Felipe Tobar. Spectral mixture kernels for multi-output gaussian processes, 2017.

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[55] Geoff Pleiss, Jacob R. Gardner, Kilian Q. Weinberger, and Andrew Gordon Wilson. Constant-time predictive distributions for gaussian processes, 2018.

[56] Geoff Pleiss, Martin Jankowiak, David Eriksson, Anil Damle, and Jacob R. Gardner. Fast matrix square roots with applications to gaussian processes and bayesian optimization, 2020.

[57] Manish Prajapat, Amon Lahr, Johannes Köhler, Andreas Krause, and Melanie N. Zeilinger. Towards safe and tractable gaussian process-based mpc: Efficient sampling within a sequential quadratic programming framework, 2024.

[58] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(65):1939–1959, 2005.

[59] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

[60] Carl Rasmussen and Zoubin Ghahramani. Occam's razor. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

[61] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005.

[62] Walter Rudin. *Fourier Analysis on Groups*. Tracts in Pure & Applied Mathematics. John Wiley & Sons, Nashville, TN, December 1962.

[63] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016.

[64] Matthias W. Seeger, Christopher K. I. Williams, and Neil D. Lawrence. Fast forward selection to speed up sparse gaussian process regression. In Christopher M. Bishop and Brendan J. Frey, editors, *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, volume R4 of *Proceedings of Machine Learning Research*, pages 254–261. PMLR, 03–06 Jan 2003. Reissued by PMLR on 01 April 2021.

[65] Paz Fink Shustin and Haim Avron. Gauss-legendre features for gaussian process regression, 2021.

[66] Harshit Sikchi, Wenxuan Zhou, and David Held. Learning off-policy with online planning, 2021.

[67] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR.

[68] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18. MIT Press, 2005.

[69] Samuel Stanton, Wesley J. Maddox, Ian Delbridge, and Andrew Gordon Wilson. Kernel interpolation for scalable online gaussian processes, 2021.

[70] Bhavya Sukhija, Lenart Treven, Cansu Sancaktar, Sebastian Blaes, Stelian Coros, and Andreas Krause. Optimistic active exploration of dynamical systems, 2023.

[71] Shengyang Sun, Daniele Calandriello, Huiyi Hu, Ang Li, and Michalis Titsias. Information-theoretic online memory selection for continual learning, 2022.

[72] Danica J. Sutherland and Jeff Schneider. On the error of random fourier features, 2015.

[73] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, July 1991.

[74] Richard S Sutton and Andrew G Barto. *Reinforcement Learning*. Adaptive Computation and Machine Learning series. Bradford Books, Cambridge, MA, 2 edition, November 2018.

[75] Abdolreza Taheri, Joni Pajarinen, and Reza Ghabcheloo. Gpu-accelerated policy optimization via batch automatic differentiation of gaussian processes for real-world control, 2022.

[76] Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In David van Dyk and Max Welling, editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 567–574, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR.

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

[78] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024.

[79] Alexandre B. Tsybakov. *Introduction to Nonparametric Estimation*. Springer New York, 2009.

[80] Cédric Villani. *Optimal Transport*. Springer Berlin Heidelberg, 2009.

[81] Jie Wang and Youmin Zhang. A tutorial on gaussian process learning-based model predictive control, 2024.

[82] Ke Alexander Wang, Geoff Pleiss, Jacob R. Gardner, Stephen Tyree, Kilian Q. Weinberger, and Andrew Gordon Wilson. Exact gaussian processes on a million data points, 2019.

[83] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning, 2019.

[84] Xiyao Wang, Junge Zhang, Wenzhen Huang, and Qiyue Yin. Planning with exploration: Addressing dynamics bottleneck in model-based reinforcement learning, 2021.

[85] Zi Wang, Clement Gehring, Pushmeet Kohli, and Stefanie Jegelka. Batched large-scale bayesian optimization in high-dimensional spaces, 2018.

[86] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, May 1992.

[87] Paweł Wawrzyński. A cat-like robot real-time learning to run. In *Adaptive and Natural Computing Algorithms*, Lecture notes in computer science, pages 380–390. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[88] Christopher Williams and Carl Rasmussen. Gaussian processes for regression. In D. Touretzky, M.C. Mozer, and M. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8. MIT Press, 1995.

[89] Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

[90] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. Model predictive path integral control using covariance variable importance sampling, 2015.

[91] Andrew Gordon Wilson and Ryan Prescott Adams. Gaussian process kernels for pattern discovery and extrapolation, 2013.

[92] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. Deep kernel learning, 2015.

[93] Andrew Gordon Wilson and Hannes Nickisch. Kernel interpolation for scalable structured gaussian processes (kiss-gp), 2015.

[94] James T. Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc Peter Deisenroth. Efficiently sampling functions from gaussian process posteriors, 2020.

[95] James T. Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc Peter Deisenroth. Pathwise conditioning of gaussian processes, 2021.

[96] Chenjun Xiao, Yifan Wu, Chen Ma, Dale Schuurmans, and Martin Müller. Learning to combat compounding-error in model-based reinforcement learning, 2019.

[97] Tianbao Yang, Yu-feng Li, Mehrdad Mahdavi, Rong Jin, and Zhi-Hua Zhou. Nyström method vs random fourier features: A theoretical and empirical comparison. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[98] Shifan Zhao, Jiaying Lu, Ji Yang, Edmond Chow, and Yuanzhe Xi. Efficient two-stage gaussian process regression via automatic kernel search and subsampling, 2024.

Appendices

A Theoretical Discussions on TD-MPC

A.1 Wasserstein Distance

For TD-MPC, all models are implemented using deterministic MLPs [33], in contrast to previous works that employ probabilistic models to mitigate the dynamics bottleneck dilemma (i.e., the relatively poor asymptotic performance compared to methods that are model-free or based on the ground-truth dynamics) [15, 39, 83, 84]. We argue that the total variation distance commonly employed in the analysis of MBRL methods (e.g., [39, 66, 84]) is not an appropriate choice for measuring the error of a deterministic model relative to the stochastic ground-truth dynamics.

The total variation distance [79] between two probability measures p and q on a measurable space $(\Omega, \mathcal{F})$ is defined by

$$D_{\mathrm{TV}}(p\|q) = \sup_{A\in\mathcal{F}} |p(A) - q(A)| \in [0, 1] \tag{A.1.1}$$

which implies $D_{\mathrm{TV}}(p\|q) = 1$ when p and q have disjoint supports, regardless of how far the supports are from each other [3]. Furthermore, if p satisfies $p(\Omega \setminus \{z_q\}) = 1$ (e.g., a Gaussian measure) and $q = \delta_{z_q}$ is deterministic at z_q (i.e., a Dirac measure), we also have

$$D_{\mathrm{TV}}(p\|q) = \sup_{A\in\mathcal{F}} |p(A) - q(A)| = 1. \tag{A.1.2}$$

.

Therefore, instead of using the total variation distance, we consider the Kantorovich-Rubinstein dual form of the Wasserstein distance, defined by

$$W(p, q) = \sup_{\|g\|_L \leq 1} \mathbb{E}_{z\sim p}[g(z)] - \mathbb{E}_{z\sim q}[g(z)] \tag{A.1.3}$$

where $\|g\|_L$ is the Lipschitz constant of a function g [80]. Then, the dynamics model error w.r.t. the Wasserstein distance could be defined by

$$\epsilon_m = \max_{s,a} W(\hat{M}(\cdot|s, a), M(\cdot|s, a)). \tag{A.1.4}$$

In contrast to the observation in (A.1.2), the Wasserstein distance enables the comparison of deterministic models with stochastic ground-truth dynamics, making it potentially suitable for the performance analysis of TD-MPC. For example, we have

$$W(p, q) = \mathbb{E}_{z\sim p}[\|z - z_q\|_2] \tag{A.1.5}$$

if $q = \delta_{z_q}$ is a Dirac measure [80].

A.2 H-step Model-based Value Error

Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, M, r, \rho_0)$ be the original MDP and π be a deterministic policy. Let Q^π denote the ground-truth action-value function of π evaluated in $\mathcal{M}$. Suppose we have a learned dynamics model $\hat{M}$, a learned reward model $\hat{r}$, and a learned action-value function $\hat{Q}^\pi$, we would like to consider two H-step finite horizon MDPs $\mathcal{H} = (\mathcal{S}, \mathcal{A}, p, r_{\text{mix}}, \rho_0)$ and $\hat{\mathcal{H}} = (\mathcal{S}, \mathcal{A}, \hat{M}, \hat{r}_{\text{mix}}, \rho_0)$ (i.e., the "simulated" MDP) where

$$r_{\text{mix}}(s_t, a_t) = \begin{cases} r(s_t, a_t) & (t < H) \\ Q^\pi(s_t, a_t) & (t = H) \end{cases} \quad \text{and} \quad \hat{r}_{\text{mix}}(s_t, a_t) = \begin{cases} \hat{r}(s_t, a_t) & (t < H) \\ \hat{Q}^\pi(s_t, a_t) & (t = H) \end{cases} \tag{A.2.1}$$

We are interested in the model-based value error $|V^\pi(s) - \hat{V}^\pi(s)|$, where the value functions $V^\pi(s)$ and $\hat{V}^\pi(s)$ represent the expected performance of policy π (starting at the initial state s) evaluated in $\mathcal{H}$ and $\hat{\mathcal{H}}$ respectively. Intuitively, this error is related to the dynamics bottleneck dilemma for MBRL [83, 84]. When this performance gap is large, the algorithm tends to select suboptimal action sequences even with a perfect optimizer for the MPC objective, causing the algorithm's performance to plateau at a level lower than their model-free counterparts or themselves with ground-truth dynamics.

Intuitively, this performance gap should relate to the following model errors

$$\begin{cases} \epsilon_m = \max_{s,a} W(\hat{M}(\cdot|s,a), M(\cdot|s,a)) & \text{(dynamics model error)} \\ \epsilon_r = \max_{s,a} |r(s,a) - \hat{r}(s,a)| & \text{(reward error)} \\ \epsilon_q = \max_{s,a} |Q^\pi(s,a) - \hat{Q}^\pi(s,a)| & \text{(Q-function error)} \end{cases} \tag{A.2.2}$$

since the two MDPs $\mathcal{H}$ and $\hat{\mathcal{H}}$ are identical when $\epsilon_m = \epsilon_r = \epsilon_q = 0$.

Lemma A.1. *Given the two MDPs $\mathcal{H}$, $\hat{\mathcal{H}}$ and the policy π aforementioned, we have*

$$\left|V^\pi(s) - \hat{V}^\pi(s)\right| \leq \underbrace{\|V^\pi\|_L \frac{\gamma - \gamma^{H+1}}{1-\gamma} \epsilon_m}_{\textit{dynamics gap}} + \underbrace{\frac{1-\gamma^H}{1-\gamma}\epsilon_r + \gamma^H \epsilon_q}_{\textit{return estimation gap}} \tag{A.2.3}$$

for any state $s \in \mathcal{S}$.

Proof. We follow the steps of Xiao et al. [96] with minor modifications. For $0 \leq h \leq H+1$, define U_h to be the H-step value expansion that, starting at the given state s, rolls out the approximate model $(\hat{M}, \hat{r}_{\text{mix}})$ first for $h-1$ steps and then rolls out the ground-true model (M, r_{mix}) for the remaining $H-h+1$ steps:

$$U_h = \sum_{t=0}^{h-1} \gamma^t \mathbb{E}_{s_t \sim \hat{P}_t^\pi(\cdot)}[\hat{r}_{\text{mix}}(s_t, \pi(s_t))] + \sum_{t=h}^{H} \gamma^t \mathbb{E}_{s_t \sim P_{t-h}^\pi \circ \hat{P}_h^\pi(\cdot)}[r_{\text{mix}}(s_t, \pi(s_t))] \tag{A.2.4}$$

where $P_t^\pi(\cdot)$ (resp. $\hat{P}_t^\pi(\cdot)$) denotes the state visitation probability at timestep t w.r.t. the ground-truth dynamics M (resp. the learned dynamics model $\hat{M}$) and a given policy π. Furthermore, $P_{t-h}^\pi \circ \hat{P}_h^\pi(\cdot) = \mathbb{E}_{s' \sim \hat{P}_h^\pi(\cdot)}[P_{t-h}^\pi(\cdot|s')]$ denotes the state visitation probability after rolling out h steps with $\hat{M}$ and then $t-h$ steps with M, where $P_h^\pi(\cdot|s')$ denotes the corresponding state visitation probability given that the roll out starts from state s'.

By some rearrangements, we have

$$\begin{aligned} U_h = &\sum_{t=0}^{h-1} \gamma^t \mathbb{E}_{s_t \sim \hat{P}_t^\pi(\cdot)}[\hat{r}(s_t, \pi(s_t))] + \gamma^h \mathbb{E}_{s_h \sim \hat{P}_h^\pi(\cdot)}[r(s_h, \pi(s_h))] \\ &+ \gamma^{h+1} \mathbb{E}_{s_{h+1} \sim P_1^\pi \circ \hat{P}_h^\pi(\cdot)}\left[V^\pi(s_{h+1})\right] \end{aligned} \tag{A.2.5}$$

$$U_{h+1} = \sum_{t=0}^{h} \gamma^t \mathbb{E}_{s_t \sim \hat{P}_t^\pi(\cdot)}[\hat{r}(s_t, \pi(s_t))] + \gamma^{h+1} \mathbb{E}_{s_{h+1} \sim \hat{P}_{h+1}^\pi(\cdot)}\left[V^\pi(s_{h+1})\right] \tag{A.2.6}$$

for $h < H$.

Therefore, for $h < H$, we can bound $U_h - U_{h+1}$ by

$$\begin{aligned} &\gamma^{h+1}\left(\mathbb{E}_{s_{h+1} \sim P_1^\pi \circ \hat{P}_h^\pi(\cdot)}\left[V^\pi(s_{h+1})\right] - \mathbb{E}_{s_{h+1} \sim \hat{P}_{h+1}^\pi(\cdot)}\left[V^\pi(s_{h+1})\right]\right) \\ &\quad + \gamma^h \mathbb{E}_{s_h \sim \hat{P}_h^\pi(\cdot)}[r(s_h, \pi(s_h)) - \hat{r}(s_h, \pi(s_h))] \\ &\leq \gamma^{h+1} \mathbb{E}_{s_h \sim \hat{P}_h^\pi(\cdot)}\left[\mathbb{E}_{s_{h+1} \sim M(\cdot|s_h, \pi(s_h))}\left[V^\pi(s_{h+1})\right] - \mathbb{E}_{s_{h+1} \sim \hat{M}(\cdot|s_h, \pi(s_h))}\left[V^\pi(s_{h+1})\right]\right] \\ &\quad + \gamma^h \epsilon_r \\ &\leq \|V^\pi\|_L \gamma^{h+1} \mathbb{E}_{s_h \sim \hat{P}_h^\pi(\cdot)}\left[W(\hat{M}(\cdot|s_h, \pi(s_h)), M(\cdot|s_h, \pi(s_h)))\right] + \gamma^h \epsilon_r \\ &\leq \|V^\pi\|_L \gamma^{h+1} \epsilon_m + \gamma^h \epsilon_r \end{aligned} \tag{A.2.7}$$

from equations (A.2.5) and (A.2.6). Moreover, for $h = H$, we have

$$\begin{aligned} U_H - U_{H+1} &= \gamma^H \left(\mathbb{E}_{s_H \sim \hat{P}_H^\pi(\cdot)}[Q^\pi(s_H, \pi(s_H))] - \mathbb{E}_{s_H \sim \hat{P}_H^\pi(\cdot)}[\hat{Q}^\pi(s_H, \pi(s_H))]\right) \\ &\leq \gamma^H \epsilon_q. \end{aligned} \tag{A.2.8}$$

By definition, $V^\pi(s) = U_0$ and $\hat{V}^\pi(s) = U_{H+1}$, hence

$$V^\pi(s) - \hat{V}^\pi(s) = \sum_{h=0}^{H} (U_h - U_{h+1}) \leq \|V^\pi\|_L \frac{\gamma - \gamma^{H+1}}{1-\gamma} \epsilon_m + \frac{1-\gamma^H}{1-\gamma} \epsilon_r + \gamma^H \epsilon_q. \tag{A.2.9}$$

By symmetry, we have

$$\left|V^\pi(s) - \hat{V}^\pi(s)\right| \leq \|V^\pi\|_L \frac{\gamma - \gamma^{H+1}}{1-\gamma} \epsilon_m + \frac{1-\gamma^H}{1-\gamma} \epsilon_r + \gamma^H \epsilon_q \tag{A.2.10}$$

which concludes the proof. □

Notice that the RHS of Lemma A.1 is dependent on the policy π, which could be arbitrary. We would like to further relax this RHS to obtain a policy-independent bound. This is achieved by making several assumptions, particularly on the Lipschitz continuity, about the original MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, M, r, \rho_0)$ and the policy class Π.

Assumption 1 (Bounded Value Function)**.** Suppose there exists a constant $V_{\max} < \infty$ such that $|V^\pi(s)| \leq V_{\max}$ for all $s \in \mathcal{S}$ and $\pi \in \Pi$.

Assumption 2 (Lipschitz Continuity)**.** The reward function r is L_R-Lipschitz continuous, and any deterministic policy $\pi \in \Pi$ is L_π-Lipschitz continuous. Furthermore, the stochastic dynamics M is L_M-Lipschitz continuous in the sense that

$$D_{\mathrm{TV}}(M(\cdot|s_1, a_1), M(\cdot|s_2, a_2)) \leq L_M \|(s_1, a_1) - (s_2, a_2)\|_2 \tag{2.2.6}$$

for all $s_1, s_2 \in \mathcal{S}$ and $a_1, a_2 \in \mathcal{A}$, where $D_{\mathrm{TV}}(\cdot, \cdot)$ denotes the total variation distance [79].

Assumptions regarding Lipschitz continuity are common in theoretical analyses of model-based reinforcement learning (e.g., [19, 70]). However, for the stochastic ground-truth dynamics M, we specifically control the changes in the output distribution using the total variation distance. We should also notice that

$$D_{\mathrm{TV}}(M(\cdot|s_1, a_1), M(\cdot|s_2, a_2)) = \frac{1}{2} \int_{s'} |M(s'|s_1, a_1) - M(s'|s_2, a_2)| ds' \tag{A.2.11}$$

for all $s_1, s_2 \in \mathcal{S}$ and $a_1, a_2 \in \mathcal{A}$ from Scheffé's theorem [79].

Remark. We made a slight abuse of notation in (A.2.11) by using $M(\cdot|s, a)$ to represent both the probability measure (on the LHS) and the probability density function (on the RHS) associated with the stochastic dynamics M conditioned on s and a.

To demonstrate that Assumption 2 makes sense, we provide a concrete example.

Example 1. Consider a GP on $\mathcal{S} \times \mathcal{A}$ with the RBF kernel and noise $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$, which should have L_m-Lipschitz continuous predictive mean $m(\cdot)$ and L_{σ^2}-Lipschitz continuous predictive variance $\sigma^2(\cdot)$ [19, 44]. Let $\mu_i = m(s_i, a_i)$ and $\sigma_i = \sigma(s_i, a_i)$ for $i = 1, 2$, we have

$$\begin{aligned} D_{\mathrm{TV}}(\mathcal{N}(\mu_1, \sigma_1^2), \mathcal{N}(\mu_2, \sigma_2^2)) &\leq \frac{3|\sigma_1^2 - \sigma_2^2|}{2\sigma_1^2} + \frac{|\mu_1 - \mu_2|}{2\sigma_1} \\ &\leq \frac{3|\sigma_1^2 - \sigma_2^2|}{2\sigma_\epsilon^2} + \frac{|\mu_1 - \mu_2|}{2\sigma_\epsilon} \\ &\leq \frac{3L_{\sigma^2} + \sigma_\epsilon L_m}{2\sigma_\epsilon^2} \|(s_1, a_1) - (s_2, a_2)\|_2 \end{aligned} \tag{A.2.12}$$

for all $s_1, s_2 \in \mathcal{S}$ and $a_1, a_2 \in \mathcal{A}$. The first inequality is from Theorem 1.3 in [22].

Lemma A.2. *Let $\mathcal{X}, \mathcal{Y}$ be Euclidean spaces and $(\mathcal{Z}, d)$ be a metric space. Given that $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ is L_f-Lipschitz continuous and $g : \mathcal{X} \to \mathcal{Y}$ is L_g-Lipschitz continuous, $h : \mathcal{X} \to \mathcal{Z}$ defined by $h(x) = f(x, g(x))$ must be $\left(L_f\sqrt{1+L_g^2}\right)$-Lipschitz continuous.*

Proof. For any x_1, x_2, we have

$$\begin{aligned} d(f(x_1, g(x_1)), f(x_2, g(x_2))) &\leq L_f \|(x_1, g(x_1)) - (x_2, g(x_2))\|_2 \\ &= L_f \sqrt{\|x_1 - x_2\|_2^2 + \|g(x_1) - g(x_2)\|_2^2} \\ &\leq L_f \sqrt{\|x_1 - x_2\|_2^2 + L_g^2 \|x_1 - x_2\|_2^2} \\ &= \left(L_f \sqrt{1 + L_g^2}\right) \|x_1 - x_2\|_2 \end{aligned} \tag{A.2.13}$$

which concludes the proof. □

Lemma A.3. *For an MDP $\mathcal{M}$ satisfying Assumption 1 and Assumption 2, we have*

$$\|V^\pi\|_L \leq (L_R + 2\gamma V_{\max} L_M)\sqrt{1 + L_\pi^2} \tag{A.2.14}$$

for any given $\pi \in \Pi$.

Proof. By the Bellman equation, we have

$$V^\pi(s) = r(s, \pi(s)) + \gamma \mathbb{E}_{s' \sim M(\cdot|s,\pi(s))}[V^\pi(s')] \tag{A.2.15}$$

for all $s \in \mathcal{S}$, which implies

$$\begin{aligned} &|V^\pi(s_1) - V^\pi(s_2)| \\ &\leq |r(s_1, \pi(s_1)) - r(s_2, \pi(s_2))| + \gamma |\mathbb{E}_{s' \sim M(\cdot|s_1,\pi(s_1))}[V^\pi(s')] - \mathbb{E}_{s' \sim M(\cdot|s_2,\pi(s_2))}[V^\pi(s')]| \\ &\leq L_R \sqrt{1 + L_\pi^2} \|s_1 - s_2\|_2 + \gamma \left| \int_{s'} (M(s'|s_1, \pi(s_1)) - M(s'|s_2, \pi(s_2))) V^\pi(s') ds' \right| \\ &\leq L_R \sqrt{1 + L_\pi^2} \|s_1 - s_2\|_2 + \gamma \int_{s'} |M(s'|s_1, \pi(s_1)) - M(s'|s_2, \pi(s_2))| \cdot |V^\pi(s')| ds' \\ &\leq L_R \sqrt{1 + L_\pi^2} \|s_1 - s_2\|_2 + \gamma V_{\max} \int_{s'} |M(s'|s_1, \pi(s_1)) - M(s'|s_2, \pi(s_2))| ds' \\ &= L_R \sqrt{1 + L_\pi^2} \|s_1 - s_2\|_2 + 2\gamma V_{\max} D_{\mathrm{TV}}(M(\cdot|s_1, \pi(s_1)), M(\cdot|s_2, \pi(s_2))) \\ &\leq L_R \sqrt{1 + L_\pi^2} \|s_1 - s_2\|_2 + 2\gamma V_{\max} L_M \sqrt{1 + L_\pi^2} \|s_1 - s_2\|_2 \\ &\leq (L_R + 2\gamma V_{\max} L_M)\sqrt{1 + L_\pi^2} \|s_1 - s_2\|_2 \end{aligned} \tag{A.2.16}$$

for all $s_1, s_2 \in \mathcal{S}$. Therefore, we have

$$\|V^\pi\|_L \leq (L_R + 2\gamma V_{\max} L_M)\sqrt{1 + L_\pi^2} \tag{A.2.17}$$

which concludes the proof. □

Theorem 1 summarizes the above discussions.

Theorem 1 (H-step Model-based Value Error)**.** *Given a policy $\pi \in \Pi$. Suppose $\hat{M}$ is a (deterministic) dynamics model such that $\max_{s,a} W(M(\cdot|s,a), \hat{M}(\cdot|s,a)) \leq \epsilon_m$, $\hat{r}$ is an approximate reward model such that $\max_{s,a} |r(s,a) - \hat{r}(s,a)| \leq \epsilon_r$, and $\hat{Q}^\pi$ is an approximate action-value function such that $\max_{s,a} |Q^\pi(s,a) - \hat{Q}^\pi(s,a)| \leq \epsilon_q$. Let V^π denote the ground-truth value function and $\hat{V}^\pi$ denote the model-based value estimation*

$$\hat{V}^\pi(s) = \mathbb{E}_{\hat{\tau} \sim \hat{P}^\pi(\cdot|s)} \left[\sum_{t=0}^{H-1} \gamma^t \hat{r}(s_t, a_t) + \gamma^H \hat{Q}(s_H, a_H) \right] \tag{2.2.7}$$

where $\hat{\tau} = (s_0, \pi(s), \ldots, s_H, \pi(s_H))$ is a trajectory sampled by $\hat{M}$ and π starting at the initial state $s_0 = s$ (i.e., $\hat{\tau} \sim \hat{P}^\pi(\cdot|s)$), then there is a constant factor $K_{\mathcal{M}}$ such that

$$\left| V^\pi(s) - \hat{V}^\pi(s) \right| \leq \underbrace{K_{\mathcal{M}} \frac{\gamma - \gamma^{H+1}}{1-\gamma} \epsilon_m}_{\textit{dynamics gap}} + \underbrace{\frac{1-\gamma^H}{1-\gamma} \epsilon_r + \gamma^H \epsilon_q}_{\textit{return estimation gap}} \tag{2.2.8}$$

for any initial state $s \in \mathcal{S}$, where $K_{\mathcal{M}}$ satisfies $K_{\mathcal{M}} \leq (L_R + 2\gamma V_{\max} L_M)\sqrt{1 + L_\pi^2}$.

A.3 Compounding Error Phenomenon

Given a deterministic policy π and an initial state s_0, we are interested in bounding $W(P_H^\pi(\cdot|s_0), \hat{P}_H^\pi(\cdot|s_0))$, i.e. the Wasserstein distance between the ground-truth dynamics M and the learned model $\hat{M}$ after H-step rollouts.

The approximate dynamics $\hat{M}$ which we compare with the ground-truth M is a single deterministic model, i.e., a special case of the class of dynamics models induced by Lipschitz model classes introduced by Asadi et al. [3]. However, we consider our policy π to be a deterministic function instead of being determined by a fixed action sequence.

Assumption 3. The learned dynamics $\hat{M}$ is $L_{\hat{M}}$-Lipschitz continuous.

Lemma A.4. *Given a deterministic policy $\pi \in \Pi$, we have*

$$W(P_h^\pi(\cdot|s_0), \hat{P}_h^\pi(\cdot|s_0)) \leq \epsilon_m \sum_{t=0}^{h-1} \left(L_{\hat{M}} \sqrt{1 + L_\pi^2} \right)^t \tag{A.3.1}$$

for any initial state $s_0 \in \mathcal{S}$ and step length $h \in \mathbb{N}$.

Proof. We follow the steps of Asadi et al. [3]. The base case when $h = 1$ is trivial. For the inductive step, we assume that

$$W(P_h^\pi(\cdot|s_0), \hat{P}_h^\pi(\cdot|s_0)) \leq \epsilon_m \sum_{t=0}^{h-1} \left(L_{\hat{M}} \sqrt{1 + L_\pi^2} \right)^t. \tag{A.3.2}$$

By the triangle inequality, $W(P^\pi_{h+1}(\cdot|s_0), \hat{P}^\pi_{h+1}(\cdot|s_0))$ is bounded by

$$W(P^\pi_{h+1}(\cdot|s_0), \hat{P}^\pi_1 \circ P^\pi_h(\cdot|s_0)) + W(\hat{P}^\pi_1 \circ P^\pi_h(\cdot|s_0), \hat{P}^\pi_{h+1}(\cdot|s_0)). \tag{A.3.3}$$

For the first term, we have

$$\begin{aligned}
&W(P^\pi_{h+1}(\cdot|s_0), \hat{P}^\pi_1 \circ P^\pi_h(\cdot|s_0)) \\
&= \sup_{\|f\|_L \le 1} \mathbb{E}_{s_{h+1}\sim P^\pi_{h+1}(\cdot|s_0)}[f(s_{h+1})] - \mathbb{E}_{s_{h+1}\sim \hat{P}^\pi_1 \circ P^\pi_h(\cdot|s_0)}[f(s_{h+1})] \\
&= \sup_{\|f\|_L \le 1} \mathbb{E}_{s_h\sim P^\pi_h(\cdot|s_0)}[\mathbb{E}_{s_{h+1}\sim M(\cdot|s_h,\pi(s_h))}[f(s_{h+1})] - \mathbb{E}_{s_{h+1}\sim \hat{M}(\cdot|s_h,\pi(s_h))}[f(s_{h+1})]] \\
&\le \mathbb{E}_{s_h\sim P^\pi_h(\cdot|s_0)}\left[\sup_{\|f\|_L \le 1} \mathbb{E}_{s_{h+1}\sim M(\cdot|s_h,\pi(s_h))}[f(s_{h+1})] - \mathbb{E}_{s_{h+1}\sim \hat{M}(\cdot|s_h,\pi(s_h))}[f(s_{h+1})]\right] \\
&\le E_{s_h\sim P^\pi_h(\cdot|s_0)}[W(M(\cdot|, s_h, \pi(s_h)), \hat{M}(\cdot|s_h, \pi(s_h))] \\
&\le \epsilon_m
\end{aligned} \tag{A.3.4}$$

For the second term, we use Lemma A.2 and the induction hypothesis,

$$\begin{aligned}
&W(\hat{P}^\pi_1 \circ P^\pi_h(\cdot|s_0), \hat{P}^\pi_{h+1}(\cdot|s_0)) \\
&= \sup_{\|f\|_L \le 1} \mathbb{E}_{s_h\sim P^\pi_h(\cdot|s_0)}[f(\hat{M}(s_h, \pi(s_h))] - \mathbb{E}_{s_h\sim \hat{P}^\pi_h(\cdot|s_0)}[f(\hat{M}(s_h, \pi(s_h))] \\
&\le L_{\hat{M}}\sqrt{1+L_\pi^2} \cdot W(P^\pi_h(\cdot|s_0), \hat{P}^\pi_h(\cdot|s_0)) \\
&\le \epsilon_m \sum_{t=1}^{h} \left(L_{\hat{M}}\sqrt{1+L_\pi^2}\right)^t
\end{aligned} \tag{A.3.5}$$

Therefore, combining the results of (A.3.4) and (A.3.5), we have

$$\begin{aligned}
W(P^\pi_{h+1}(\cdot|s_0), \hat{P}^\pi_{h+1}(\cdot|s_0)) &\le \epsilon_m + \epsilon_m \sum_{t=1}^{h} \left(L_{\hat{M}}\sqrt{1+L_\pi^2}\right)^t \\
&\le \epsilon_m \sum_{t=0}^{h} \left(L_{\hat{M}}\sqrt{1+L_\pi^2}\right)^t
\end{aligned} \tag{A.3.6}$$

which completes the proof by mathematical induction. □

We obtain a corollary, which is also a special case of Theorem 1 in Asadi et al. [3].

Corollary A.5. *Given a policy π determined by a fixed action sequence, we have*

$$W(P^\pi_h(\cdot|s_0), \hat{P}^\pi_h(\cdot|s_0)) \le \epsilon_m \sum_{t=0}^{h-1} \left(L_{\hat{M}}\right)^t \tag{A.3.7}$$

for any initial state $s_0 \in \mathcal{S}$ and step length $h \in \mathbb{N}$.

B Scalable Gaussian Process Regression

B.1 More on the Mathematical Formulations

In Section 2.3, we have briefly introduced the function-space view of GPs. Alternatively, we could also interpret GP as kernel Bayesian linear regression (i.e., the weight-space view) [61]. Let $\phi(\cdot)$ be the feature map corresponding to the kernel function $k(\cdot, \cdot)$ (i.e., $k(\mathbf{x}_1, \mathbf{x}_2) = \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle$), and consider the linear model $y = \phi(\mathbf{x})^\top \mathbf{w} + \varepsilon$. If the prior of $\mathbf{w}$ is $p(\mathbf{w}) \sim \mathcal{N}(0, \mathbf{I})$, then the posterior of $\mathbf{w}$ becomes

$$p(\mathbf{w} \,|\, X, \mathbf{y}) \propto p(\mathbf{y} \,|\, X, \mathbf{w}) p(\mathbf{w}) \propto \exp\left(-\frac{1}{2} (\mathbf{w} - \bar{\mathbf{w}})^\top A (\mathbf{w} - \bar{\mathbf{w}}) \right) \tag{B.1.1}$$

where $A = \sigma_\epsilon^{-2} \phi(X) \phi(X)^\top + \mathbf{I}$ and $\bar{\mathbf{w}} = \sigma_\epsilon^{-2} A^{-1} \phi(X) \mathbf{y}$. Hence, the predictive distribution

$$p(f(\mathbf{x}^*) \,|\, X, \mathbf{y}, \mathbf{x}^*) = \int p(f(\mathbf{x}^*) \,|\, \mathbf{x}^*, \mathbf{w}) p(\mathbf{w} \,|\, X, \mathbf{y}) \, d\mathbf{w} \tag{B.1.2}$$

is equivalent to (2.3.1) by direct computation [61].

Remark. For the weight-space view, the feature map corresponding to a kernel matrix does not necessarily map data points to a finite-dimensional space as assumed above. We take this assumption for an easier illustration of the underlying intuition.

The information gain for the GP model with inputs X most commonly refers to the mutual information [18] between observations $\mathbf{y}$ and values of $\hat{f}$ on X respectively, i.e.

$$\mathrm{IG}(\mathbf{y}; \hat{f}) = \frac{1}{2} \log |\mathbf{I} + \sigma_\epsilon^{-2} K_{XX}| = \frac{1}{2} \log |\hat{K}_{XX}| - n \log \sigma_\epsilon \tag{B.1.3}$$

which reflects the model complexity that appeared in the training objective (2.3.5): lower information gains indicate less detailed accounts of observations $\mathbf{y}_X$ in predictions $\hat{f}$, meaning the model is simpler, and vice versa. In the context of reinforcement learning, the notion of this information gain could be used to design intrinsic rewards to guide optimistic exploration of the unknown environment [70].

Another information gain of interest in the context of continual learning is the mutual information between the new data point $(\mathbf{x}^*, y^*)$ and the current model [64, 71]. For Gaussian process models, we consider its weight-space view

$$\begin{aligned}
\mathrm{IG}((\mathbf{x}^*, y^*); p(\mathbf{w} \,|\, X, \mathbf{y})) &= D_{\mathrm{KL}}(p(\mathbf{w} \,|\, X, \mathbf{y}, \mathbf{x}^*, y^*) \,|\, p(\mathbf{w} \,|\, X, \mathbf{y})) \\
&= \mathbb{E}_{p(\mathbf{w} \,|\, X, \mathbf{y}, \mathbf{x}^*, y^*)}[\log p(y^* \,|\, \mathbf{w}, \mathbf{x}^*)] - \log p(y^* \,|\, X, \mathbf{y}, \mathbf{x}^*) \\
&\leq \log \mathbb{E}_{p(\mathbf{w} \,|\, X, \mathbf{y}, \mathbf{x}^*, y^*)}[p(y^* \,|\, \mathbf{w}, \mathbf{x}^*)] - \log p(y^* \,|\, X, \mathbf{y}, \mathbf{x}^*) \\
&= \underbrace{\log p(y^{**} \,|\, X, \mathbf{y}, \mathbf{x}^*, y^*)|_{y^{**} = y^*}}_{\text{learnability}} - \underbrace{\log p(y^* \,|\, X, \mathbf{y}, \mathbf{x}^*)}_{\text{surprise}}
\end{aligned} \tag{B.1.4}$$

which has an upper bound in the closed form [71].

B.2 Kernel Approximation Methods

Referring to equations (2.3.1), (2.3.5), and (2.3.6), the training and inference of GPs involve computing the matrix solves $\hat{K}_{XX}^{-1}\mathbf{y}$ and $\hat{K}_{XX}^{-1}\mathbf{k}_{X\mathbf{x}^*}$, the determinant term $|\hat{K}_{XX}|$, and the trace term $\operatorname{tr}\left(\hat{K}_{XX}^{-1}\frac{\partial \hat{K}_{XX}}{\partial \theta}\right)$, which are the main computational bottlenecks.

In general, by finding a low-rank approximation of the $n \times n$ kernel matrix K_{XX}, we could apply the Woodbury identity to approximate $\hat{K}_{XX}^{-1} = (K_{XX}^{-1} + \sigma_\epsilon^2 \mathbf{I})^{-1}$ in $O(n^2 s)$ time, where $s \ll n$ is the rank of the approximation. Furthermore, it only takes $O(ns^2)$ time for matrix solves by arranging the order of computations. In this subsection, we briefly overview some common low-rank kernel approximation methods for GPs, including Fourier features (e.g., random Fourier features [59]) and Nyström approximation [89], as well as structured kernel interpolation (SKI) [93].

B.2.1 Fourier Features

The stationary kernels mentioned in Section 2.3 admit Fourier feature approximations by Bochner's theorem (2.3.3) [59, 62]. More specifically, s independent samples of $\{\omega_i\}_{i=1}^s$ from the spectral density $p(\omega)$ and $\{b_i\}_{i=1}^s$ from $\text{Unif}([0, 2\pi])$ can be used to approximate the kernel function via

$$k(\mathbf{x}, \mathbf{y}) = \mathbb{E}_\omega[\xi_\omega(\mathbf{x}_1)\xi_\omega(\mathbf{x}_2)^*] \approx \frac{1}{s}\sum_{i=1}^{s} \sqrt{2}\cos(\omega_i^\top \mathbf{x} + b_i) \cdot \sqrt{2}\cos(\omega_i^\top \mathbf{y} + b_i). \tag{B.2.1}$$

Let $\mathbf{z}(\mathbf{x}) = \frac{1}{\sqrt{s}}(\sqrt{2}\cos(\omega_1^\top \mathbf{x} + b_1), \ldots, \sqrt{2}\cos(\omega_s^\top \mathbf{x} + b_s))^\top \in \mathbb{R}^s$, we can approximate the kernel matrix $K_{XX} \approx Z_X^\top Z_X$ where $Z_X = [\mathbf{z}(\mathbf{x}_1); \ldots; \mathbf{z}(\mathbf{x}_n)]^\top \in \mathbb{R}^{s\times n}$.[11] This approximation has rank at most s. Furthermore, we could interpret this approximation from the weight-space view, considering $\mathbf{z}(\cdot)$ as an "approximation" of the feature map $\phi(\cdot)$. In this case, the posterior distribution (B.1.1) of the weight $\mathbf{w} \in \mathbb{R}^s$ should be

$$p(\mathbf{w} \mid X, \mathbf{y}) = \mathcal{N}(\sigma_\epsilon^{-2}\tilde{A}^{-1}Z_X\mathbf{y}, \tilde{A}) \tag{B.2.2}$$

where $\tilde{A} = \sigma_\epsilon^{-2}Z_X Z_X^\top + \mathbf{I} \in \mathbb{R}^{s\times s}$. To sample from the posterior, it takes $O(s^2 n)$ to compute $\tilde{A}$, $O(s^3)$ time to compute the Cholesky decomposition of $\tilde{A}$, and $O(s^2 n)$ to compute $\tilde{A}^{-1}Z_X$ from the Cholesky decomposition.[12] Notice that the sampled posterior function $\mathbf{w}^\top \mathbf{z}(\cdot)$ is deterministic and only takes $O(s)$ time for evaluation.

[11] The embedding $\frac{1}{\sqrt{s}}(\sqrt{2}\cos(\omega_1^\top \mathbf{x}), \sqrt{2}\sin(\omega_1^\top \mathbf{x}), \ldots, \sqrt{2}\cos(\omega_{s/2}^\top \mathbf{x}), \sqrt{2}\sin(\omega_{s/2}^\top \mathbf{x}))^\top$ could be used as an alternative. In fact, the latter always produces lower variances for approximating the RBF kernel [72], and is used in the GPyTorch implementation of RFF kernels [28].

[12] Furthermore, posterior sampling from equation (B.2.2) could be computed efficiently via contour integral quadrature (CIQ) [56] in $O(Jns)$ time (typically, $J \leq 100$ is the number of iterations). CIQ is an MVM-based method, which is amenable to GPU acceleration.

However, RFF approximations suffer from variance starvation, producing unreliable confidence bounds [52, 85]. Mutný et al. propose the quadratic Fourier feature (QFF) [52], which takes the perspective of numerical integration instead of Monte Carlo approximation. Generally, they use the quadrature rule, essentially a weighted sum of function evaluations (known as integration points). Different integration points and corresponding weights could be selected by applying different quadrature rules, such as the Gauss-Hermite quadrature (in QFF), the Gauss-Legendre quadrature [65], and the trigonometric quadrature [45]. If we restrict to the 1-D scenario, these deterministic Fourier feature methods require significantly fewer feature dimensions to attain acceptable approximations compared to the RFF approach, and are less prone to variance starvation; However, to extend them to higher dimensions, a Cartesian product grid is required, which grows exponentially with the number of dimensions D [45, 52, 65]. Consequently, certain measures need to be taken to amend the CoD (e.g., Mutný et al. propose to use generalized additive models [52]).

Alternatively, Avron et al. [4] propose to use quasi-Monte Carlo (QMC) estimation to replace the Monte Carlo estimation in the original approach of RFF [59]. Although QMC obtains a convergence rate of $O((\log s)^D/s)$ which also suffers the curse of dimensionality, it has been empirically shown that QMC still out-performs MC even in high-dimensional settings [4]. Intuitively, quasi-uniform sampling tends to avoid the undesired clustering.

B.2.2 Nyström Approximation

The Nyström approximation, also referred to as the subset of regressors (SoR), is also a common technique for low-rank kernel matrix approximation [89]. Specifically, partition the kernel matrix K_{XX} into block matrices $K_{s,s} \in \mathbb{R}^{s\times s}$, $K_{s,n-s} = K_{n-s,s}^\top \in \mathbb{R}^{s\times(n-s)}$, and $K_{n-s,n-s} \in \mathbb{R}^{(n-s)\times(n-s)}$ as follows

$$K_{XX} = \begin{bmatrix} K_{s,s} & K_{s,n-s} \\ K_{n-s,s} & K_{n-s,n-s} \end{bmatrix} \tag{B.2.3}$$

then a rank-s approximation of K_{XX} is obtained by

$$K_{XX} \approx K_{n,s} K_{s,s}^{-1} K_{s,n} \tag{B.2.4}$$

in $O(ns^2)$ time[13], where $K_{n,s} = [K_{s,s}; K_{n-s,s}] \in \mathbb{R}^{n\times s}$ and $K_{s,n} = K_{n,s}^\top \in \mathbb{R}^{s\times n}$. The benefit of this approach is that it is data-dependent, hence it is considered theoretically and empirically better in certain scenarios compared to the Fourier features approach [97].

The intuition behind the Nyström approximation could be understood from the projection perspective [51]. Let $S \in \mathbb{R}^{n\times s}$ be a one-hot matrix such that each column has a single non-zero entry equals 1 so that $K_{XX}S = K_{n,s}$ and $S^\top K_{XX} S = K_{s,s}$. Recall that

[13]The Cholesky decomposition of $K_{s,s}$ requires $O(s^3)$ time, which is negligible when $s \ll n$.

$K_{XX} = \phi(X)^\top \phi(X)$ where $\phi(X) \in \mathbb{R}^{m\times n}$ is the feature embedding of the data set (assume $m < \infty$ is the feature dimension). We consider the projection of the feature embedding $\phi(X)$ to the column space of $\phi(X)S \in \mathbb{R}^{m\times s}$, which is the feature embedding of a subset of the dataset. The corresponding projection matrix P is given by

$$P = \phi(X)S(S^\top \phi(X)^\top \phi(X)S)^{-1}S^\top \phi(X)^\top \in \mathbb{R}^{m\times m} \tag{B.2.5}$$

which indeed satisfies $P^2 = P$ and is symmetric (i.e., $P^\top = P$). Consider the projection $P\phi(X) \in \mathbb{R}^{m\times n}$ as an approximation of the feature embedding $\phi(X)$, then we obtain an approximation of the kernel matrix

$$\begin{aligned} K_{XX} &= \phi(X)^\top \phi(X) \\ &\approx \phi(X)^\top P^\top P\phi(X) \\ &= \phi(X)^\top \phi(X)S(S^\top \phi(X)^\top \phi(X)S)^{-1}S^\top \phi(X)^\top \phi(X) \\ &= K_{n,s}K_{s,s}^{-1}K_{s,n}. \end{aligned} \tag{B.2.6}$$

Remark. The partition in (B.2.3) we consider in the Nyström approximation could be arbitrary. Intuitively, a "good" approximation should ensure that the space spanned by the projected feature vectors $\mathrm{col}(\phi(X)S)$ is of the highest dimensionality (up to $\dim(\mathrm{col}(\phi(X)))$. To achieve this, the subset of points used to construct $K_{s,s}$, which are known as the landmark points, should be distributed as evenly as possible [11].

The structured kernel interpolation (SKI) [93] is an extension of the Nyström approximation, which enables an efficient full-rank approximation by leveraging a structured set of inducing points (or landmark points for Nyström approximation). Specifically, SKI uses a structured grid (e.g., Toeplitz or Kronecker) as inducing points to form $K_{s,s}$, and builds an $O(n)$-sparse matrix $W \in \mathbb{R}^{n\times s}$ via cubic interpolation [40] so that $K_{n,s} \approx WK_{s,s}$. Plugging them into equation (B.2.4), we have

$$K_{XX} \approx K_{n,s}K_{s,s}^{-1}K_{s,n} = WK_{s,s}K_{s,s}^{-1}K_{s,s}W = WK_{s,s}W = K_{SKI} \tag{B.2.7}$$

where it is possible to have $s \gg n$, making the approximate GP more likely to maintain exact performance. However, the efficiency of matrix-vector multiplications (MVM) with K_{SKI} is significant, requiring at most $O(n + s\log s)$ time. This efficiency arises from the structured nature of $K_{s,s}$ and the sparsity of W, which enable fast approximations of kernel matrix solves using black-box MVM methods such as conjugate gradients. As a result, SKI facilitates scalable training while maintaining computational feasibility. Furthermore, the SKI-based approach allows for constant-time mean and variance predictions during inference through Lanczos decomposition and caching [55].

However, a key limitation of SKI—similar to deterministic Fourier feature approaches [45, 52, 65]—is its restriction to low-dimensional problems (typically, $D < 5$) due to exponentially growing memory requirements [93]. Common strategies to mitigate this issue include additive kernels [23], product kernels (SKIP) [29], or deep kernel learning.

B.3 Variational Bayesian Methods

Other than directly approximating the kernel matrix K_{XX}, an alternative approach is to use variational Bayesian methods to approximate the GP posterior. Specifically, let $\mathbf{f} = f(X)$ be the prior conditioned on X where $f \sim \mathcal{GP}(0, k(\cdot,\cdot))$, and consider m inducing points $Z \in \mathcal{X}$ with $\mathbf{u} = f(Z)$. Replacing the conditional prior $p(\mathbf{u})$ with the variational distribution $q(\mathbf{u})$, we could approximate the augmented conditional prior $p(\mathbf{f}, \mathbf{u}) = p(\mathbf{f} \mid \mathbf{u})p(\mathbf{u})$ by $q(\mathbf{f}, \mathbf{u}) = p(\mathbf{f} \mid \mathbf{u})q(\mathbf{u})$ [76], which allows us to derive the evidence lower bound (ELBO) of the MLL in equation (2.3.5)

$$\begin{aligned}
\log p(\mathbf{y}) &= \log \mathbb{E}_{p(\mathbf{f},\mathbf{u})}[p(\mathbf{y} \mid \mathbf{f})] \\
&= \log \mathbb{E}_{q(\mathbf{f},\mathbf{u})}\left[\frac{p(\mathbf{y} \mid \mathbf{f})p(\mathbf{f},\mathbf{u})}{q(\mathbf{f},\mathbf{u})}\right] \\
&\geq \mathbb{E}_{q(\mathbf{f},\mathbf{u})}\left[\log \frac{p(\mathbf{y} \mid \mathbf{f})p(\mathbf{f},\mathbf{u})}{q(\mathbf{f},\mathbf{u})}\right] \\
&= \mathbb{E}_{q(\mathbf{f},\mathbf{u})}\left[\log \frac{p(\mathbf{y} \mid \mathbf{f})\cancel{p(\mathbf{f} \mid \mathbf{u})}p(\mathbf{u})}{\cancel{p(\mathbf{f} \mid \mathbf{u})}q(\mathbf{u})}\right] \\
&= \mathbb{E}_{q(\mathbf{f},\mathbf{u})}[\log p(\mathbf{y} \mid \mathbf{f})] - D_{\mathrm{KL}}(q(\mathbf{u}) \| p(\mathbf{u})).
\end{aligned} \tag{B.3.1}$$

Remark. For simplicity, we have slightly abused notation by omitting the conditioning on X and θ in this subsection. In other words, whenever we write expressions such as $p(\mathbf{y})$, we actually mean the likelihood $p(\mathbf{y} \mid X, \theta)$, unless otherwise mentioned.

Notice that we could usually obtain the factorization $\log p(\mathbf{y} \mid \mathbf{f}) = \log \prod_{i=1}^n p(\mathbf{y}_i \mid \mathbf{f}_i) = \sum_{i=1}^n \log p(\mathbf{y}_i \mid \mathbf{f}_i)$, so that the ELBO (B.3.1) admits stochastic gradients during training. However, under certain conditions (e.g., $q(\mathbf{u}) = \mathcal{N}(\mathbf{m}_\mathbf{u}, S_\mathbf{u})$ is a free Gaussian distribution), there is a closed-form optimal variational distribution that maximizes the ELBO using the calculus of variations and Lagrange multipliers. This allows the variational conditioning strategy used by Maddox et al. [47].

Lemma B.1. *The optimal variational distribution $q(\mathbf{u})$ that maximizes the ELBO*

$$\mathbb{E}_{q(\mathbf{f},\mathbf{u})}\left[\log \frac{p(\mathbf{y} \mid \mathbf{f})p(\mathbf{u})}{q(\mathbf{u})}\right] \tag{B.3.2}$$

satisfies

$$q(\mathbf{u}) \propto \exp\left(\int p(\mathbf{f} \mid \mathbf{u}) \log p(\mathbf{y} \mid \mathbf{f}) d\mathbf{f}\right) p(\mathbf{u}). \tag{B.3.3}$$

Proof. Consider the functional

$$\begin{aligned}
S[q] &= \mathbb{E}_{q(\mathbf{f},\mathbf{u})}\left[\log \frac{p(\mathbf{y} \mid \mathbf{f})p(\mathbf{u})}{q(\mathbf{u})}\right] - \eta_0\left(\int q(\mathbf{u}) d\mathbf{u} - 1\right) \\
&= \int q(\mathbf{u})\left[\int p(\mathbf{f} \mid \mathbf{u}) \log p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{u}) d\mathbf{f}\right] d\mathbf{u} + H(q) - \eta_0\left(\int q(\mathbf{u}) d\mathbf{u} - 1\right)
\end{aligned} \tag{B.3.4}$$

where η_0 is the Lagrange multiplier that ensures the second axiom of probability. We know that $S[q]$ attains an extremum when the functional derivative is equal to zero, i.e.

$$\frac{\delta S}{\delta q(\mathbf{u})} = \int p(\mathbf{f} \mid \mathbf{u}) \log(p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{u})) d\mathbf{f} + \underbrace{(-\log q(\mathbf{u}) - 1)}_{\frac{\delta H}{\delta q(\mathbf{u})}} - \eta_0 = 0 \tag{B.3.5}$$

which implies that

$$\begin{aligned} q(\mathbf{u}) &= \exp\left(\int p(\mathbf{f} \mid \mathbf{u}) \log(p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{u})) d\mathbf{f} - 1 - \eta_0 \right) \\ &\propto \exp\left(\int p(\mathbf{f} \mid \mathbf{u}) \log p(\mathbf{y} \mid \mathbf{f}) d\mathbf{f} + \log p(\mathbf{u}) \right) \\ &\propto \exp\left(\int p(\mathbf{f} \mid \mathbf{u}) \log p(\mathbf{y} \mid \mathbf{f}) d\mathbf{f} \right) p(\mathbf{u}) \end{aligned} \tag{B.3.6}$$

which concludes the proof. □

Corollary B.2. *Assuming that we have a Gaussian likelihood $p(\mathbf{y} \mid \mathbf{f}) = \mathcal{N}(\mathbf{f}, \sigma_\epsilon^2 \mathbf{I})$ and the variational distribution $q(\mathbf{u}) = \mathcal{N}(\mathbf{m}_\mathbf{u}, S_\mathbf{u})$ is Gaussian with free parameters $(\mathbf{m}_\mathbf{u}, S_\mathbf{u})$, then the optimal variational distribution w.r.t. the dataset $(X, \mathbf{y})$ satisfy*

$$\mathbf{m}_\mathbf{u} = K_{ZZ}(K_{ZZ} + C)^{-1}\mathbf{c}, \qquad S_\mathbf{u} = K_{ZZ}(K_{ZZ} + C)^{-1} K_{ZZ} \tag{B.3.7}$$

where

$$\mathbf{c} = K_{ZX}\Sigma_\mathbf{y}^{-1}\mathbf{y} \in \mathbb{R}^m, \qquad C = K_{ZX}\Sigma_\mathbf{y}^{-1}K_{XZ} \in \mathbb{R}^{m \times m} \tag{B.3.8}$$

where $\Sigma_\mathbf{y} = \sigma_\epsilon^2 \mathbf{I}$.

Proof. From the Matheron's rule [1], $p(\mathbf{f} \mid \mathbf{u}) = \mathcal{N}(K_{XZ}K_{ZZ}^{-1}\mathbf{u}, K_{XX} - K_{XZ}K_{ZZ}^{-1}K_{ZX})$. Furthermore, from $p(\mathbf{y} \mid \mathbf{f}) = \mathcal{N}(\mathbf{f}, \sigma_\epsilon^2 \mathbf{I})$, we have

$$\begin{aligned} &\int p(\mathbf{f} \mid \mathbf{u}) \log p(\mathbf{y} \mid \mathbf{f}) d\mathbf{f} \\ &= -\frac{1}{2\sigma_\epsilon^2}\mathbb{E}[\|\mathbf{y} - \mathbf{f}\|^2 \mid \mathbf{u}] + O(1) \\ &= -\frac{1}{2\sigma_\epsilon^2}(\mathbb{E}[\mathbf{f}^\top \mathbf{f} \mid \mathbf{u}] - 2\mathbb{E}[\mathbf{f} \mid \mathbf{u}]^\top \mathbf{y} + \cancel{\mathbf{y}^\top \mathbf{y}}) + O(1) \\ &= \frac{1}{2\sigma_\epsilon^2}(\mathrm{Var}(\mathbf{f}|\mathbf{u}) + \mathbb{E}[\mathbf{f} \mid \mathbf{u}]^\top \mathbb{E}[\mathbf{f} \mid \mathbf{u}] - 2\mathbb{E}[\mathbf{f} \mid \mathbf{u}]^\top \mathbf{y}) + O(1) \\ &= -\frac{1}{2\sigma_\epsilon^2}(\cancel{\mathrm{Tr}(K_{XX} - K_{XZ}K_{ZZ}^{-1}K_{ZX})} + \mathbf{u}^\top K_{ZZ}^{-1}K_{ZX}K_{XZ}K_{ZZ}^{-1}\mathbf{u} - 2\mathbf{u}^\top K_{ZZ}^{-1}K_{ZX}\mathbf{y}) + O(1) \\ &= -\frac{1}{2}\mathbf{u}^\top K_{ZZ}^{-1}CK_{ZZ}^{-1}\mathbf{u} + \mathbf{u}^\top K_{ZZ}^{-1}\mathbf{c} + O(1). \end{aligned} \tag{B.3.9}$$

By completing the square, we may verify that

$$\begin{aligned} q(\mathbf{u}) &\propto \exp\left(\int p(\mathbf{f}\,|\,\mathbf{u}) \log p(\mathbf{y}\,|\,\mathbf{f}) d\mathbf{f}\right) p(\mathbf{u}) \\ &\propto \exp\left(-\frac{1}{2}\mathbf{u}^\top K_{ZZ}^{-1} C K_{ZZ}^{-1}\mathbf{u} + \mathbf{u}^\top K_{ZZ}^{-1}\mathbf{c} - \frac{1}{2}\mathbf{u}^\top K_{ZZ}^{-1}\mathbf{u}\right) \\ &\propto \exp\left(-\frac{1}{2}\mathbf{u}^\top [K_{ZZ}(K_{ZZ}+C)^{-1}K_{ZZ}]^{-1}\mathbf{u} + \mathbf{u}^\top K_{ZZ}^{-1}\mathbf{c}\right) \\ &\propto \exp\left(-\frac{1}{2}(\mathbf{u}-\mathbf{m}_\mathbf{u})^\top S_\mathbf{u}^{-1}(\mathbf{u}-\mathbf{m}_\mathbf{u})\right) \end{aligned} \tag{B.3.10}$$

where $S_\mathbf{u} = K_{ZZ}(K_{ZZ}+C)^{-1}K_{ZZ}$ and $\mathbf{m}_\mathbf{u} = S_\mathbf{u}K_{ZZ}^{-1}\mathbf{c} = K_{ZZ}(K_{ZZ}+C)^{-1}\mathbf{c}$. □

Corollary B.2 demonstrates that variational conditioning can be computed in $O(nm^2)$ time and $O(nm)$ space, which is comparable to the complexity of solving low-rank kernel approximations. Moreover, the inference of SVGP only requires $O(m)$ time for mean prediction and $O(m^2)$ time for variance prediction after pre-computation, which is independent of the size of the dataset. Specifically, we have

$$\begin{aligned} \mathbb{E}[f(\mathbf{x}^*)\,|\,Z,\mathbf{x}^*] &= \mathbb{E}[\mathbb{E}[f(\mathbf{x}^*)\,|\,Z,\mathbf{x}^*,\mathbf{u}]] \\ &= \mathbb{E}[\mathbf{k}_{Z\mathbf{x}^*}^\top K_{ZZ}^{-1}\mathbf{u}] \\ &= \mathbf{k}_{Z\mathbf{x}^*}^\top K_{ZZ}^{-1}\mathbf{m}_\mathbf{u} \end{aligned} \tag{B.3.11}$$

and

$$\begin{aligned} \mathrm{Var}[f(\mathbf{x}^*)\,|\,Z,\mathbf{x}^*] &= \mathbb{E}[\mathrm{Var}[f(\mathbf{x}^*)\,|\,Z,\mathbf{x}^*,\mathbf{u}]] + \mathrm{Var}[\mathbb{E}[f(\mathbf{x}^*)\,|\,Z,\mathbf{x}^*,\mathbf{u}]] \\ &= k(\mathbf{x}^*,\mathbf{x}^*) - \mathbf{k}_{Z\mathbf{x}^*}^\top K_{ZZ}^{-1}\mathbf{k}_{Z\mathbf{x}^*} + \mathrm{Var}[\mathbf{k}_{Z\mathbf{x}^*}^\top K_{ZZ}^{-1}\mathbf{u}] \\ &= k(\mathbf{x}^*,\mathbf{x}^*) - \mathbf{k}_{Z\mathbf{x}^*}^\top K_{ZZ}^{-1}\mathbf{k}_{Z\mathbf{x}^*} + \mathbf{k}_{Z\mathbf{x}^*}^\top K_{ZZ}^{-1} S_\mathbf{u} K_{ZZ}^{-1}\mathbf{k}_{Z\mathbf{x}^*} \\ &= k(\mathbf{x}^*,\mathbf{x}^*) - \mathbf{k}_{Z\mathbf{x}^*}^\top K_{ZZ}^{-1}(K_{ZZ}-S_\mathbf{u})K_{ZZ}^{-1}\mathbf{k}_{Z\mathbf{x}^*}. \end{aligned} \tag{B.3.12}$$

B.4 Determinantal Point Processes

In this subsection, we briefly describe the determinantal point processes (DPP) for inducing point selection. We first consider the so-called L-ensemble representation of a DPP $\mathbb{P}_L$ on X [7]. Let $Y \in 2^X$ be a random draw according to $\mathbb{P}_L$ such that

$$\mathbb{P}_L(Y) \propto |\hat{K}_{YY}| \tag{B.4.1}$$

where the normalizing factor is $|\hat{K}_{XX}+\mathbf{I}| = \sum_{Y\in 2^X}|\hat{K}_{YY}|$. Recall equation (B.1.3), the determinant $|\hat{K}_{YY}|$ is related to the mutual information between the observations and the GP posterior evaluated at Y. Notice that in the GP training objective (2.3.5), models

with smaller information gains are preferred, which could be interpreted as an automatic Occam's Razor in Bayesian modeling [60]. However, for IPA, we would like to choose a subset of data points that correspond to high information gains.

Furthermore, let $T = (\hat{K}_{XX} + \mathbf{I})^{-1}\hat{K}_{XX}$ (consider it as a "normalized" kernel matrix), for every subset $A \subseteq X$, we have

$$\mathbb{P}(A \subseteq Y) = |T_{AA}|. \tag{B.4.2}$$

One observation from equation (B.4.2) is that, for $A = \{\mathbf{x}_1, \mathbf{x}_2\}$, let $S \in \{0,1\}^{2\times n}$ be the one-hot matrix such that $T_{AA} = STS^\top$ and $\hat{K}_{AA} = S\hat{K}_{XX}S^\top$, then

$$\mathbb{P}(\mathbf{x}_1, \mathbf{x}_2 \in Y) = |S||(\hat{K}_{XX} + \mathbf{I})^{-1}\hat{K}_{XX}||S^\top| \propto |S\hat{K}_{XX}S^\top| = |\hat{K}_{AA}| \tag{B.4.3}$$

which implies that

$$\begin{aligned}\mathbb{P}(\mathbf{x}_1, \mathbf{x}_2 \in Y) &\propto \begin{vmatrix} k(\mathbf{x}_1, \mathbf{x}_1) + \sigma_\epsilon^2 & k(\mathbf{x}_1, \mathbf{x}_2) \\ k(\mathbf{x}_1, \mathbf{x}_2) & k(\mathbf{x}_2, \mathbf{x}_2) + \sigma_\epsilon^2 \end{vmatrix} \\ &= (k(\mathbf{x}_1, \mathbf{x}_1) + \sigma_\epsilon^2)(k(\mathbf{x}_2, \mathbf{x}_2) + \sigma_\epsilon^2) - k(\mathbf{x}_1, \mathbf{x}_2)^2.\end{aligned} \tag{B.4.4}$$

That is to say, more correlated points tend to be less likely to co-occur in a sample from the DDP. Similarly, if we extend to scenarios with heteroscedastic noises, then data points associated with random noises of higher variances are more likely to be contained in the sample Y. In other words, DDP models the diversity of subsamples [42].

However, sampling from a DDP (which samples a subset of an undetermined size) requires the eigendecomposition of the kernel matrix $\hat{K}_{XX}$ in $O(n^3)$ time where n is the size of the dataset, which is typically intractable [38]. Kulesza et al. [42] propose the k-DDP approach which attempts to reduce computation by conditioning on $|Y| = k$. However, only the subsequent sampling steps are reduced to $O(nk^2)$ time, and the $O(n^3)$ eigendecomposition remains the computational bottleneck. Alternatively, Burt et al. [10] suggest approximating the *maximum a posteriori* (MAP) estimate of a k-DDP, which could be greedily obtained in $O(nk^2)$ time [14].