Exploration via Planning for Information about the Optimal Trajectory

Viraj Mehta[1], **Ian Char**[2], **Joseph Abbate**[4], **Rory Conlin**[4], **Mark D. Boyer**[3], **Stefano Ermon**[5], **Jeff Schneider**[1], **Willie Neiswanger**[5]

[1]Robotics Institute, [2]Machine Learning Department, Carnegie Mellon University
[3]Princeton Plasma Physics Laboratory, [4]Princeton University
[5]Computer Science Department, Stanford University

{virajm,ichar,schneide}@cs.cmu.edu, {jabbate,wconlin}@princeton.edu, mboyer@pppl.gov, {ermon,neiswanger}@cs.stanford.edu

Abstract

Many potential applications of reinforcement learning (RL) are stymied by the large numbers of samples required to learn an effective policy. This is especially true when applying RL to real-world control tasks, e.g. in the sciences or robotics, where executing a policy in the environment is costly. In popular RL algorithms, agents typically explore either by adding stochasticity to a reward-maximizing policy or by attempting to gather maximal information about environment dynamics without taking the given task into account. In this work, we develop a method that allows us to plan for exploration while taking both the task and the current knowledge about the dynamics into account. The key insight to our approach is to plan an action sequence that maximizes the expected information gain about the optimal trajectory for the task at hand. We demonstrate that our method learns strong policies with 2x fewer samples than strong exploration baselines and 200x fewer samples than model free methods on a diverse set of low-to-medium dimensional control tasks in both the open-loop and closed-loop control settings.[1]

1 Introduction

The potential of reinforcement learning (RL) as a general-purpose method of learning solutions to sequential decision making problems is difficult to overstate. Ideally, RL could allow for agents that learn to accomplish all manner of tasks solely through a given reward function and the agent's experience; however, RL has so far broadly fallen short of this. Chief among the difficulties in realizing this potential is the fact that typical RL methods in continuous problems require very large numbers of samples to achieve a near-optimal policy. For many interesting applications of RL this problem is exacerbated by the fact that collecting samples from the true environment can incur huge costs. For example, expensive scientific experiments are needed for tokamak control [31, 13, 20] and design of molecules [70], and collecting experience on the road is both costly and runs the risk of car accidents for autonomous vehicles [66].

Though model-based methods like Deisenroth and Rasmussen [21], Chua et al. [15], and Curi et al. [16] are much more efficient than typical model-free methods, they do not explicitly reason about the prospective task-relevant information content of future observations. Moreover, these methods typically require thousands of timesteps of environment dynamics data to solve reasonably simple Markov decision processes (MDPs). Though progress has been made in using Thompson sampling [46], entropy bonuses [27], and upper confidence bounds (UCB) [16, 5] to more intelligently explore

[1]Code is available at: https://github.com/fusion-ml/trajectory-information-rl

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

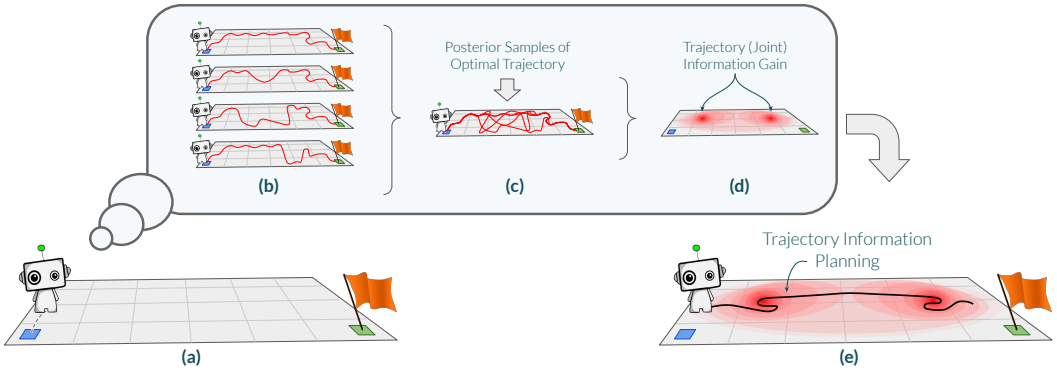

Figure 1: A schematic depiction of Trajectory Information Planning (TIP). Suppose the agent in (a) aims to determine where to explore next from its current state. To do so, in (b) the agent samples dynamics models $T' \sim P(T \mid D)$ from its current posterior and finds approximately optimal trajectories $\tau^* \sim P(\tau^* \mid T')$ for each sample. Then in (c) it pools these samples of posterior optimal trajectories τ^*. In (d) it constructs a function that gives the joint expected information gain about the optimal trajectory τ^* given a planned exploration trajectory (i.e. EIG_{τ^*} over the set of points visited). Finally, in (e) the agent can plan an action sequence which maximizes this joint expected information gain.

the state-action space of an MDP, these methods still do not explicitly reason about how information that the agent gathers will affect the estimated MDP solution. Furthermore, in continuous state-action settings these methods must make coarse approximations to be computationally tractable (e.g. bootstrapped Q networks to approxiate a posterior or one-step perturbations for approximate UCB).

Many methods in the vein of Pathak et al. [47] and Shyam et al. [61] explore directly to gather new information based on current uncertainty about environment dynamics, but they do not in general specialize the information that they aim to acquire for a particular *task* specified by an initial state distribution and reward function. We believe that an ideal exploration strategy for sample efficiency should take into account this task, as well as uncertainty about the environment dynamics.

In this work, we start by showing how many methods can be cast as a Bayesian planning problem over a specific cost function. This framework helps illuminate the importance of the cost function and what impact it has on exploration. Viewed in this light, it becomes clear that many previous state-of-the-art methods rely on cost functions that either result in behavior that is too greedy—i.e. the policy tries to maximize returns during exploration—or too exploratory, i.e. the policy is incentivized to explore the environment dynamics and does not consider the task at hand. We therefore present a cost function that balances out these two extremes, by generalizing an information-gain acquisition function introduced in Mehta et al. [40] to apply to a set of future hypothetically acquired data. In particular, our cost function captures the amount of information that would be gained about the *optimal trajectory*, if the agent were to explore by following a particular planned trajectory. As depicted in Figure 1, this involves the agent sampling what it would hypothetically do given different realizations of the dynamics, and then planning actions that are informative about those possibilities.

In summary, the contributions of this work are as follows: we develop a novel cost function for exploration that explicitly accounts for both the specific task and uncertainty about environment dynamics, a method for planning which applies the cost function to explore in MDPs with continuous states and actions, and a thorough empirical evaluation of our method across 5 closed-loop and 3 open-loop environments (with a focus on expensive RL tasks in plasma physics) compared against 14 baselines. We find that our proposed method is able to learn policies that perform as well as an agent with access to the ground truth dynamics using half or fewer samples than comparison methods.

2 Related Work

Exploration in Reinforcement Learning The most common strategy for exploration in RL is to execute a greedy policy with some form of added stochasticity. The simplest approach, ϵ-greedy exploration as used in Mnih et al. [41], takes the current action thought to be best with probability $1 - \epsilon$ and a random action with probability ϵ. Other methods use added Ornstein-Uhlenbeck action noise [37] to the greedy policy, or entropy bonuses [27] to the policy or value function objectives to add noise to a policy which is otherwise optimizing the RL objective.

Tabular RL is often solved by choosing actions based on upper confidence bounds on the value function [14, 36], but explicitly computing and optimizing these bounds in the continuous setting is substantially more challenging. Recent work [16] approximates this method by computing one-step confidence bounds on the dynamics and training a 'hallucinated' policy which chooses perturbations within these bound to maximize expected policy performance. Another recent work [5] uses anti-concentration inequalities to approximate upper confidence bounds in MDPs with discrete actions.

Thompson sampling (TS) [55], which samples a realization of the MDP from the posterior and acts optimally as if the realization was the true model, can be applied for exploration in a model-free manner as in [45] or in a model-based manner as in [63]. As the posterior over MDP dynamics or value functions can be high-dimensional and difficult to represent, the performance of TS can be hindered by approximation errors using both Gaussian processes and ensembles of neural networks. Curi et al. [16] recently investigated this and found that this was potentially due to an insufficiently expressive posterior over entire transition functions, implying that it may be quite difficult to solve tasks using sampled models. Similarly, the posterior over action-value functions in Osband et al. [45] is only roughly approximated by training a bootstrapped ensemble of neural networks.

There is also a rich literature of Bayesian methods for exploration, which are typically computationally expensive and hard to use, though they have attractive theoretical properties. These methods build upon the fundamental idea of the Bayes-adaptive MDP [53], which we detail in Section E.1 alongside a discussion of this literature.

Additionally, a broad set of methods explore to learn about the environment without addressing a specified task. This line of work is characterized by Pathak et al. [47], which synthesizes a task-agnostic reward function from model errors. Other techniques include MAX [61], which optimizes the information gain about the environment dynamics, Random Network Distillation [11], which forces the agent to learn about a random neural network across the state space, and Plan2Explore [60], which prospectively plans to find areas of novelty where the dynamics are uncertain.

Bayesian Experimental Design: BOED, BO, BAX, and BARL There is a large literature on Bayesian optimal experiment design (BOED) [12] which focuses on efficiently querying a process or function to get maximal information about some quantity of interest. When the quantity of interest is the location of a function optimum, related strategies have been proposed as the entropy search family of Bayesian optimization (BO) algorithms [29, 30]. Recently, a flexible framework known as Bayesian algorithm execution (BAX) [43] has been proposed to efficiently estimate properties of expensive black-box functions; this framework gives a general procedure for sampling points which are informative about the future execution of a given algorithm that computes the property of interest, thereby allowing the function property to be estimated with far less data.

A subsequent related work [40], known as Bayesian Active Reinforcement Learning (BARL), uses ideas from BOED and BAX to sample points that are maximally informative about the optimal trajectory in an MDP. However, BARL relies on a setting the authors call Transition Query Reinforcement Learning (TQRL), which assumes that the environment dynamics can be iteratively queried at an arbitrary sequence of state-action pairs chosen by the agent. TQRL is thus a highly restrictive setting which is not suitable when data can only be accessed via a trajectory (rollout) of environment dynamics; it typically relies on an accurate environment simulator of sufficient expense to warrant its use. Even then, there will likely be differences between simulators and ground truth dynamics for complex systems. Thus, one would ideally like to collect data in real environments. However, this often requires leaving the TQRL setting, and instead collecting data via trajectories only.

In this paper, we aim to apply the information-theoretic ideas from BARL but generalize them to the general MDP setting as well as learn open loop model-based controllers. The typical method for learning to solve open-loop control problems was demonstrated successfully in Tesch et al. [65], where a value function was learned from action sequences to task success. Our method takes a model-based approach to this problem, using similar exploration strategies as Bayesian optimization but benefitting from the more substantial supervision that is typical in dynamics model learning.

3 Problem Setting

In this work we deal with finite-horizon discrete-time *Markov decision processes* (MDPs) which consist of a sextuple $\langle \mathcal{S}, \mathcal{A}, T, r, p_0, H \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, T is the transition function $T : \mathcal{S} \times \mathcal{A} \to P(\mathcal{S})$ (using the convention that $P(\mathcal{X})$ is the set of probability measures over $\mathcal{X}$), $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a reward function, $p_0(s)$ is a distribution over $\mathcal{S}$ of start

states, and $H \in \mathbb{N}$ is the horizon (i.e. the length of an episode). We always assume $\mathcal{S}, \mathcal{A}, p_0$, and H are known. We also assume the reward r is known, though our development of the method can easily be generalized to the case where r is unknown. Our primary object of interest is the transition function T, which we learn from data. We address both open and closed loop control settings. In the more common closed loop setting, our aim is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes Objective (1) below. We will denote trajectories as $\tau \sim p(\tau \mid \pi, T)$ where $\tau = [(s_0, a_0), \ldots, (s_{H-1}, a_{H-1}), s_H]$ generated by $s_0 \sim p_0$, $a_i = \pi(s_i)$, and $s_{i+1} \sim T(s_i, a_i)$. We can write the return of a trajectory as $R(\tau) = \sum_{i=0}^{H-1} r(s_i, a_i, s_{i+1})$ for the states and actions s_i, a_i that make up τ. The MDP objective can then be written as

$$J_T(\pi) = \mathbb{E}_{\tau \sim p(\tau \mid \pi, T)} \left[R(\tau) \right]. \tag{1}$$

We aim to maximize this objective while minimizing the number of samples from the ground truth transition function T that are required to reach good performance. We denote the optimal policy as $\pi^* = \arg\max_\pi J_T(\pi)$, which we can assume to be deterministic [64] but not necessarily unique. We use τ^* to denote optimal trajectories, i.e. $\tau^* \sim p(\tau \mid \pi^*, T)$.

Similarly, for the open-loop setting, we assume a fixed start state s_0 and aim to find an action sequence $a_0, \ldots, a_{H-1}$ that maximizes the sum of rewards in an episode. We will slightly abuse notation and write $\tau \sim p(\tau \mid a_{0:H-1})$ and $J_T(a_{0:H-1})$ with these actions fixed in place of a reactive policy, and again use τ^* to refer to the trajectories generated by an optimal action sequence.

We assume in this work that applying planning algorithms like [51] to a dynamics function T will result in a trajectory that approximates τ^*. We will primarily focus on a Gaussian process (GP) model of the transition dynamics in order to take advantage of its expressive representation of uncertainty and grounded methods for sampling, conditioning, and joint inference. There is substantial prior work using GPs in RL—see Section E.2 for a discussion of this literature. Under this modeling choice, we assume that the dynamics are drawn from a GP prior $P(T)$ (see Section A.3 for further details on our GP model) and use $P(T \mid D)$ for the posterior over transition functions given a dynamics dataset of triples $D = \{(s_i, a_i, s_i')\}$. In this work, unions $D \cup \tau$ or $D \cup S'$ between the dataset D and trajectories τ or next state predictions S' coerce τ and S' into triples of dynamics data, prior to the union with the dataset.

4 Trajectory Information Planning

Our method consists of a generic framework for Bayesian (or approximately Bayesian) model-predictive control and a novel cost function for planning that allows us to explicitly plan to find the maximal amount of new information relevant to our task. In Section 4.1, we describe the MPC framework and highlight that many prior methods approximate this framework while using a greedy cost function that corresponds to the future negative expected rewards or a pure exploration cost function that corresponds to future information about the dynamics. Afterwards, in Section 4.2, we derive our new cost function and describe how it is computed. The overall method we introduce simply applies this planning framework with our new cost function.

4.1 Model-Predictive Control in Bayesian Model-Based RL

In this section, we give a formulation of Bayesian planning for control that generalizes ideas from methods such as PILCO [21] and PETS [15]. This formulation highlights these methods' inherently greedy nature and hints at a possible solution. The objective of Bayesian planning is to find the h-step action sequence that maximizes the expected future returns under model uncertainty. That is,

$$\operatorname*{argmin}_{a_0, \ldots, a_{h-1}} \mathbb{E}_{T' \sim P(T \mid D), \tau_e \sim P(\tau \mid s_0 = s, a_{0:h-1}, T')} \left[C(\tau_e) \right] \tag{2}$$

for some cost function C over trajectories and some start state s. If operating in the open-loop control setting, the agent executes the sequence of actions found without replanning. This procedure can also be extended to closed-loop control via model-predictive control (MPC), which involves re-planning (2) at every state the agent visits and playing the first action from the optimal sequence. Concretely, the MPC policy for our Bayesian setting is as follows:

$$\pi_{\text{MPC}}(s) = \arg\min_{a_0} \min_{a_1, \ldots, a_{h-1}} \mathbb{E}_{T' \sim P(T \mid D), \tau_e \sim P(\tau \mid s_0 = s, a_{0:h-1}, T')} \left[C(\tau_e) \right] \tag{3}$$

Whether we do open-loop control or closed-loop control via MPC, the cost function C, is integral to how the agent will behave. Prior work has predominantly focused on two types of cost function:

$$\underbrace{C_g(\tau) = -R(\tau)}_{\text{Greedy Exploration}} \qquad \underbrace{C_e(\tau) = -\sum_{i=0}^{h} \mathbb{H}\left[T(s_i, a_i) \mid D\right]}_{\text{Task-Agnostic Exploration}} \tag{4}$$

Previous works such as Kamthe and Deisenroth [34] and PETS [15] use the greedy exploration cost function, C_g. This cost function incentivizes trajectories that achieve high rewards over the next h transitions on average. In works that focus on task-agnostic exploration such as Sekar et al. [60] and Shyam et al. [61], the cost function C_e (or similar) is used to encourage the agent to find areas of the state space in which the model is maximally uncertain. Note that we use π_g to refer to the greedy policy given by using (3) with C_g.

The optimization problem in (3) is typically approximately solved in one of three ways: Deisenroth and Rasmussen [21] and Curi et al. [16] directly backpropagate through the estimated dynamics and reward functions to find policy parameters that would generate good actions, Janner et al. [32] use an actor-critic method trained via rollouts in the model alongside the data collected to find a policy, and Chua et al. [15] and Mehta et al. [40] use the cross-entropy method [17] to find action sequences which directly maximize the reward over the estimated dynamics. In this work, we use a version of the last method given in Pinneri et al. [51], denoted iCEM, to directly find action sequences that optimize the cost function being used. We approximate the expectation by playing the actions on multiple samples from the posterior $P(T \mid D)$. Algorithm 1 gives a formal description of the method and Section A.5 provides further details.

Algorithm 1 Bayesian Model-Predictive Control with Cost Function C

Inputs: transition function episode query budget b, number of posterior function samples k, planning horizon h.
Initialize $D \leftarrow \emptyset$.
for $i \in [1, \ldots, b]$ **do**
  Sample start state $s_0 \sim p_0$.
  for $t \in [0, \ldots, H-1]$ **do**
    Sample posterior functions $\{T'_\ell\}_{\ell=1}^{k} \sim P(T' \mid D)$.
    Approximately find $\arg\min_{a_0,\ldots,a_{h-1}} \sum_{\ell=1}^{k} \mathbb{E}_{\tau_\ell \sim p(\tau \mid T'_\ell, a_0, \ldots, a_{h-1})}\left[C(\tau_\ell)\right]$ via iCEM.
    Execute action a_0 by sampling $s_{t+1} \sim T(s_t, a_0)$.
    Update dataset $D \leftarrow D \cup \{(s_t, a_0, s_{t+1}\}$.
  end for
end for
return π_g for the posterior $P(T' \mid D)$.

4.2 A Task-Specific Cost Function based on Trajectory Information

In this work, we aim to explore by choosing actions that maximize the conditional expected information gain (EIG) about the optimal trajectory τ^*. This is the same overall goal as that of Mehta et al. [40], where the EIG_{τ^*} acquisition function was introduced for this purpose. However, in this paper we generalize this acquisition function in order to allow for sequential information collection, and account for the redundant information that could be collected between timesteps. As discussed at length in Osband et al. [46], it is essential to reason about how an action taken at the current timestep will affect the possibility of learning something useful in future timesteps. In other words, exploration must be *deep* and not greedy. Explicit examples are given in Osband et al. [46] where the time to find an ϵ-optimal policy in a tabular MDP is exponential in the state size unless exploration can be coordinated over large numbers of timesteps rather than being conducted independently at each action. As the EIG_{τ^*} acquisition function is only defined over a single state-action pair and mutual information is submodular, we cannot naively use the acquisition function as is (or sum it over many datapoints) to choose actions that lead to good long-term exploration. This is clear in e.g. navigation tasks, where the nearby points visited over trajectories will provide redundant information about the local environment.

We therefore give a cost function that generalizes EIG_{τ^*} by taking a set of points to query and computing the *joint* expected information gain from observing the set. Our cost function is non-

Markovian in the state space of the MDP, but it is Markovian in the dataset, which represents a point in the belief space of the agent about the dynamics. Let $\mathcal{X} = \{x : x \subseteq \mathcal{S} \times \mathcal{A}, |x| < \infty\}$ be the set of finite subsets of the set of all state-action pairs. Our cost function $C_{\tau^*} : \mathcal{X} \to \mathbb{R}$ is defined below to be the negative *joint expected information gain* about the optimal trajectory τ^* for a subset $X \in \mathcal{X}$. In particular, assuming an existing dataset D, a set of h query points $X = \{(s_i, a_i)\}_{i \in [h]}$, and a random set of next states $S' = \{s'_i \sim T(s_i, a_i), i \in [h]\}$,

$$C_{\tau^*}(X) = \mathbb{E}_{S' \sim p(S'|X,D)} \left[\mathbb{H}\left[\tau^* \mid D \cup S'\right]\right] - \mathbb{H}\left[\tau^* \mid D\right]. \tag{5}$$

This formulation of C_{τ^*} forces our method to handle the redundant information among queries—it is likely that $I(s'_1, \tau^*) + I(s'_2, \tau^*) > I(\{s'_1, s'_2\}, \tau^*)$ and our method should avoid this overestimation. However, as written, this function relies on computing entropies on high-dimensional trajectories where the form of the joint distribution of the elements is unknown. To tractably estimate this quantity, we use the fact that $C_{\tau^*}(X) = -I(S', \tau^*) = -I(\tau^*, S')$ for the mutual information I. This allows us to exchange τ^* and our set of queries so that τ^* is giving information about the posterior predictive distribution of our set. In other words,

$$C_{\tau^*}(X) = \mathbb{E}_{\tau^* \sim p(\tau^*|D)} \left[\mathbb{H}\left[S' \mid D \cup \tau^*\right]\right] - \mathbb{H}\left[S' \mid D\right]. \tag{6}$$

In order to compute the right-hand term, we must take samples $\tau^*_{ij} \sim P(\tau^* \mid D), i = 1, \ldots, m, j = 1, \ldots, n$. To do this, we first sample m start states $s_0^{(i)}$ from p_0 (we always set $m = 1$ in experiments but derive the procedure in general) and for each start state independently sample n posterior functions $T'_{ij} \sim P(T' \mid D)$ from our posterior over dynamics models. We then run a planning procedure using iCEM [51] on each of the posterior functions from $s_0^{(i)}$ using T'_{ij} for T (using our assumption that planning can generate approximately optimal trajectories given ground-truth dynamics), giving our sampled τ^*_{ij}. Formally, we can approximate C_{τ^*} via Monte-Carlo as

$$C_{\tau^*}(X) \approx \frac{1}{mn} \left(\sum_{i \in [m]} \sum_{j \in [n]} \mathbb{H}[S' | D \cup \tau^*_{ij}] \right) - \mathbb{H}[S' \mid D]. \tag{7}$$

Assuming the dynamics are modelled with a Gaussian process, we can compute the joint Gaussian probability of the next states S' [52]. As the entropy of a multivariate Gaussian depends only on the log-determinant of the covariance, $\log|\Sigma|$, we can tractably compute the joint entropy of the model predictions $\mathbb{H}\left[S' \mid D\right]$ and optimize it with a zeroth order optimization algorithm. Finally, we must calculate the entropy $\mathbb{H}[S' | D \cup \tau^*_{ij}]$. For this, we follow a similar strategy as Neiswanger et al. [43]: since τ^*_i is a set of states given by the transition model, we can treat them as additional noiseless datapoints for our dynamics model and condition on them before computing the joint covariance matrix for S'. Given this newly generalized acquisition function, we can instantiate a method of planning in order to maximize future information gained. We give the concrete procedure for computing our acquisition function in Algorithm 2, noting that trajectories τ^*_{ij} do not depend on the query set X and can be cached for various values of X as long as the dataset D does not change.

Our ultimate procedure, which we name *Trajectory Information Planning* (TIP), is quite simple: run model-based RL using MPC as in Algorithm 1, but set the cost function to be $C_{\tau^*}(\tau)$ instead of C_g or C_e, and compute this cost function using Algorithm 2. At test time, we return to planning with C_g as the cost function and greedily attempt to maximize returns rather than performing exploration. We can also formulate an open-loop variant of our method, oTIP, which involves planning once and then executing the entire action sequence.

4.3 Computational Cost and Implementation Details

Though the TIP algorithm is designed for settings where samples are expensive, it is important to understand, both theoretically and practically, the computational cost of this method. For ease of notation, we make the simplifying assumption that the planning algorithm used (in this case, iCEM from [51]) evaluates p action sequences consisting of h (the planning horizon) actions and that our current dataset is of size N. In order to efficiently sample functions from the posterior over dynamics functions, we use the method from Wilson et al. [69]. This reduces the naive complexity of querying these functions from $O(N^3)$ to a one-time $O(N)$ cost and then $O(1)$ for additional queries. As we derive in Section A.1, the computational complexity of one TIP planning iteration is $O\left(nm\left((N+H)^3 + ph\,(N+H)^2\right)\right)$. The two asymptotically expensive operations are (1)

Algorithm 2 Computation of C_{τ^*}

Inputs: dataset $D = \{(s_k, a_k, s'_k)\}$, query set X, number of start state samples m, number of posterior function samples n.

Sample m start states $\{s_0^{(i)}\}_{i=1}^m \sim p_0$.
for $i \in [m]$ **do**
  Sample n posterior functions $\{T'_j\}_{j=1}^n \sim P(T' \mid D)$.
  for $j \in [n]$ **do**
    Set $\pi_j^* \leftarrow \pi_{\text{MPC}}$ using C_g and a singleton posterior $P(T \mid D) = \delta(T'_j)$ as in (3).
    Compute τ_{ij}^* by executing π_j^* on T'_j starting from $s_0^{(i)}$.
  end for
end for
Compute the joint posterior covariance $\Sigma^{S'} \mid D$ across all points in X.
Compute the joint posterior covariances $\Sigma_{ij}^{S'} \mid D \cup \tau_{ij}\, \forall i \in [n], j \in [m]$ across all points in X.
return $\log|\Sigma^{S'}| - \frac{1}{nm}\sum_{i\in[n],j\in[m]} \log|\Sigma_{ij}^{S'}|$.

computing the Cholesky decompositions of the nm kernel matrices for datasets $D \cup \tau_{ij}^*$ and (2) solving the triangular systems using the cached Cholesky decompositions in order to compute the covariance matrices $\Sigma_{ij}^{S'} \mid D \cup \tau_{ij}^*$ for each of the p action sequences used by the planning algorithm.

However, our implementation choices mean that in practice these operations are not the most expensive step. The covariance matrix computations, which are the theoretical bottleneck, are implemented in JAX [9], allowing them to be compiled to much faster machine code and vectorized across large batches of queries. In fact, the most expensive operation in practice is planning on the sampled transition functions T'_i to sample optimal trajectories τ_{ij}^*. This is due to the fact that in practice p is large and we implemented the planner in NumPy [28] so it cannot be compiled together with the Tensorflow [1] code from Wilson et al. [69], which is used for predicting which states will be visited for the planner. We give further information on the implementation in Section A.

5 Experiments

The aim of our development of the TIP algorithm and the C_{τ^*} acquisition function for RL is to reduce the sample complexity of learning effective policies in continuous MDPs given limited access to expensive dynamics. In this section we demonstrate the effectiveness of TIP in quickly learning a good policy by comparing against a variety of state-of-the-art reinforcement learning algorithms and strong baselines (including some that use the TQRL setting from [40], which is also known as RL with a generative model in Kakade [33] and other works [6, 4]).

In particular, we compare the average return across five evaluation episodes across five random seeds of each algorithm on five closed-loop control problems. For sample complexity we assess the median amount of data taken by each algorithm to 'solve' the problem across five seeds with the threshold performance given by an MPC controller using the ground truth dynamics. We evaluate the open-loop variant of our method, oTIP, against three comparison methods on three control problems suitable for open-loop control. In particular, to be suitable for open-loop control, the problem cannot be dynamically unstable (as Pendulum and Cartpole famously are) and must have a relatively short control horizon and fixed start state.

	Greedy (C_g) w/ Stochasticity	Task-agnostic (C_e) Exploration	Task-specific (C_{τ^*}) Exploration
TQRL	N/A	EIG$_T$	BARL
Rollout (Sum)	MPC, PETS, BPTT, PPO, SAC, TD3	sDIP	sTIP
Rollout (Joint)		DIP	TIP, TS, HUCRL

Figure 2: Our comparison methods can be broken down by the type of cost function used and how the methods do or do not handle sequential acquisition of information. As C_g is a sum, it naturally handles future timesteps jointly. For the other information quantities, it is possible to upper-bound information acquired by summing each separate mutual information, or to compute them jointly.

Environment	TIP	sTIP	DIP	MPC	PETS	SAC	FEEF	RHC	HUCRL	TS	BARL	EIG_T
Pendulum	**21**	36	36	46	5.6k	7k	800	>40k	>50k	>50k	**21**	56
Cartpole	131	141	161	201	1.63k	32k	>2.5k	>5k	>6k	>6k	**111**	121
β Tracking	**46**	76	276	76	330	12k	300	>3k	480	420	186	>1k
β + Rotation	**201**	>500	>500	>500	400	30k	>2k	>2k	>5k	>5k	>500	>1k
Reacher	**251**	>400	>1k	751	700	23k	>5k	1.5k	6.6k	4.5k	**251**	>1.5k

Table 1: **Sample Complexity:** Median number of samples across five seeds required to reach 'solved' performance, averaged across five trials. We determine 'solved' performance by running an MPC policy (similar to the one used for evaluation) on the ground truth dynamics to predict actions. We record $> n$ when the median run is unable to solve the problem by the end of training after collecting n datapoints. The methods in the rightmost section operate in the TQRL setting and therefore have more flexible access to the MDP dynamics for data collection. The full set of methods are shown in Section D as well as boxplots depicting the data in Figure 4.

Here too, we assess the average return as open-loop trials are conducted as well as the number of timesteps required to achieve 'solved' performance.

Comparison Methods We use several model-based and model-free comparison methods in this work. We compare to several published model-based methods. These include **PETS** [15], as implemented by Pineda et al. [50], which uses a probabilistic ensemble of neural networks and CEM over particle samples to do MPC. We also compare against three model-based techniques from the HUCRL [16] implementation: **HUCRL** itself, which relies on hallucinating dynamics perturbations as a way of realizing an upper confidence bound on the policy, model-based Thompson Sampling (**TS**), which samples from the posterior over models and chooses optimal actions for that sample, and a greedy model-based neural network method relying on backpropagation through time (**BPTT**). We also compare against the Free Energy of the Expected Future method from Tschantz et al. [67], which treats directed exploration as a process of actively collecting information for inference on a reward-biased generative model. A further comparison is with Receding Horizon Curiosity (RHC) [59], which does online Bayesian system identification over a linear model in order to quickly find a model of the environment dynamics. Our model-free comparison methods, Soft Actor-Critic (**SAC**) [27], an actor-critic method that uses an entropy bonus over the policy to encourage more exploration, and two others (TD3 and PPO), are in the appendix.

Finally, we compare against various ablations of the proposed method. These vary across two axes as described in Figure 2: the cost function they use and how they handle sequential queries. Besides these differences, they use the same GP model and iCEM planning algorithm with the same hyperparameters, so they are truly comparable methods. The three cost functions used are C_g, C_{τ^*}, and C_e. **DIP**, **oDIP**, and **EIG**$_T$ all use C_e but compute, respectively, the expected joint entropy of the action sequence, the sum of the pointwise entropies of the action sequence, and the individual pointwise entropies in the TQRL setting. These methods are very similar in spirit to [61, 48] in that they plan for future information gain about the dynamics, but we chose to compare in a way that controls for difference in the model and planning algorithm. **MPC** uses C_g and is very close to the method in [34]. Like TIP, **BARL** [40] and **sTIP** use C_{τ^*}. BARL operates in the TQRL setting and can therefore use the simpler EIG_{τ^*} acquisition function. sTIP investigates the use of $-\sum_{s_i,a_i \in S} \text{EIG}_{\tau^*}(s_i, a_i)$ as a cost function for planning. This computes individual information gains for each future observation without accounting for the information they may have in common and is therefore an overestimate of the joint information gain. In the open-loop setting we compare against **oDIP** and **oMPC**, the open-loop variants of DIP and MPC, and Bayesian optimization (**BO**) as implemented by Pedregosa et al. [49]. oDIP plans an action sequence to minimize the joint C_e and executes the actions found for each open-loop trial, while oMPC does the same thing using C_g. We give additional details on the comparison methods in Section B.

Control Problems Our closed loop control problems are the standard underactuated **Pendulum** swing-up task (Pendulum-v0 from Brockman et al. [10]) with 2D states and 1D actions, a **Cartpole** swing-up task with a sparse reward function, 2D s, and 1D actions, a 2-DOF robot arm control problem where the end effector is moved to a goal (**Reacher**-v2 from Brockman et al. [10]) with 10D states and 2D actions, a simplified β **Tracking** problem from plasma control [13, 39] (similar in design but not identical to the one from Mehta et al. [40]) trained using with 4D states and 1D actions, and a more complicated problem in plasma control where β **+ Rotation** are tracked with 10D states and 2D actions. Our open loop control problems are a navigation problem with hazards (**Lava Path**) from [40] and two regulation problems with different **Nonlinear Gain** functions. The Lava Path problem has 4D states and 2D actions and the nonlinear regulation problems have 2D states and 2D actions. Full details on these problems are available in Section C.

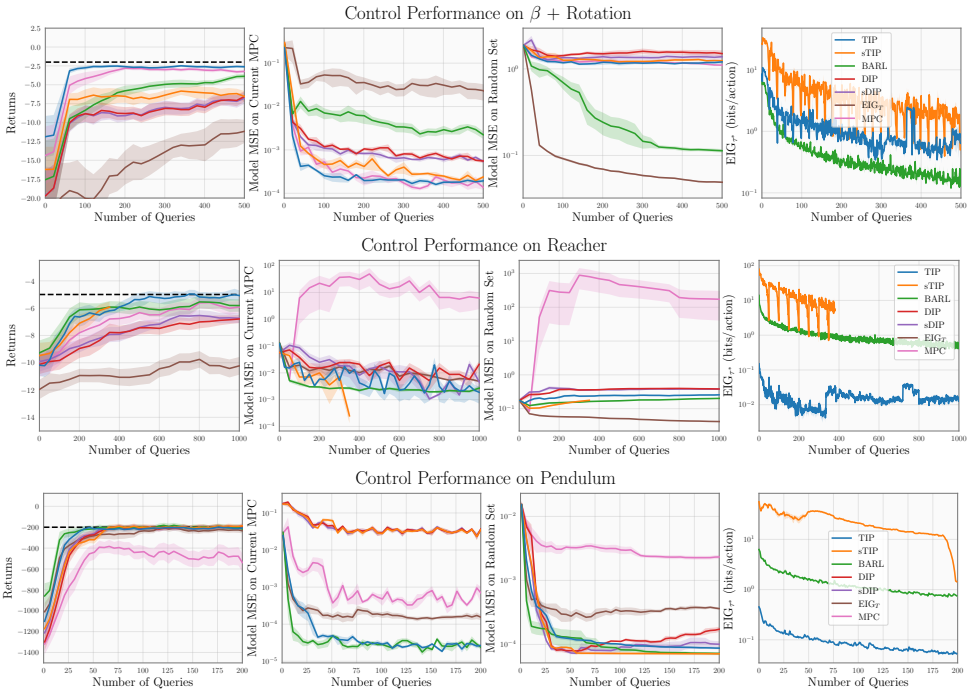

Figure 3: **Control and Modeling Details for TIP and Ablations.** Column 1: Learning curves for our ablation methods, all of which use the same planner and model. Column 2: Dynamics model accuracy on the points used by the planner to choose actions during MPC. Column 3: Dynamics model accuracy on a uniformly random test set in $\tilde{\mathcal{S}}$. Column 4: EIG_{τ^*} values normalized by the number of actions planned. sTIP was truncated on Reacher as it exceeded the wall time budget.

Results As can be seen in Table 1, TIP is able to reach solved performance more quickly across the board than the model-based and model-free external baselines, often using a fraction or even orders of magnitude less data than other methods. For many of our ablation methods we see failures to solve some of the problems even though the model is demonstrated by TIP to be able to sufficiently predict the dynamics. This is especially apparent on the harder plasma control environment, β+Rotation, where TIP is the only method using our GP which is able to solve the problem. We believe that this is because the data acquired through exploration by the ablation methods is less useful for control than the data TIP collects. This is underscored by the second column of Figure 3, where it is clear that TIP achieves the lowest modeling error on the points actually needed during the execution of the policy but not on the uniform test set. In particular we find it interesting that TIP outperforms BARL on the β + Rotation environment, as BARL should in principle have a strictly stronger access to the problem and is optimizing the same quantity with fewer constraints. We hypothesis that this may be due to the fact that BARL optimizes the acquisition function EIG_{τ^*} by simply uniformly sampling a set of points and choosing the one that evaluates to the largest value. Our more sophisticated optimization algorithm and forced initialization at the start state distribution seems to allow us to collect more information in this case. This interpretation is bolstered by the fact that on the problems where TIP outperforms BARL, we see that TIP is actually collecting more information per action than BARL as evidenced by larger EIG values. We also see clearly that there is value in computing the C_{τ^*} function rather than summing over EIG_{τ^*} values, as TIP outperforms sTIP across the board. Additionally, there is clear evidence for the value of task-specific exploration as the task-agnostic exploration methods (EIG_T, DIP, sDIP) underperform both in returns and model error on the trajectories visited.

Environment	oTIP	oMPC	oDIP	BO
Nonlinear Gain 1	**41**	91	51	210
Nonlinear Gain 2	**51**	61	>200	60
Lava Path	**41**	101	101	>2k

Table 2: **Open Loop Sample Complexity:** Median number of samples required to reach 'solved' performance, averaged across five trials. We determine 'solved' performance by running an MPC policy on the ground truth dynamics to predict actions. We record $> n$ when the median run is unable to solve the problem by the end of training after collecting n datapoints.

For the open-loop experiments (Table 2), we also see strong performance from oTIP. As the model-based methods benefit from observing many model transitions for each open-loop trial, it is unsurprising that they are more sample-efficient than the BO method. Within the model-based techniques, oTIP is the most sample efficient. We believe that this is for much the same reasons as in the closed-loop case—exciting evidence that the C_{τ^*} cost function can be applied in a variety of settings.

6 Conclusion

In this work, we presented and evaluated a cost function designed for intelligent, task-aware exploration. Using this cost function in model-predictive control allows agents to solve continuous MDPs with far less data than comparison methods in open- and closed-loop settings. Though the method is effective in data reduction, it is computationally expensive and relies on dynamics that are well-modeled by a GP. In future work, we aim to scale the method to higher-dimensional and more complex control environments. We also aim to apply this method in the real world. In particular, we aim to address similar plasma control problems in a small number of trials on a real tokamak.

Acknowledgements

This work was supported in part by US Department of Energy grants under contract numbers DE-SC0021414 and DE-AC02-09CH1146.

This work is supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE1745016 and DGE2140739. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

This work is additionally supported by NSF (#1651565), AFOSR (FA95501910024), ARO (W911NF-21-1-0125), CZ Biohub, and Sloan Fellowship.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Joseph Abbate, R Conlin, and E Kolemen. Data-driven profile prediction for diii-d. *Nuclear Fusion*, 61(4):046027, 2021.

[3] Jan Achterhold and Joerg Stueckler. Explore the context: Optimal data collection for context-conditional dynamics models. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3529–3537. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/achterhold21a.html.

[4] Alekh Agarwal, Sham Kakade, and Lin F. Yang. Model-based reinforcement learning with a generative model is minimax optimal. In Jacob Abernethy and Shivani Agarwal, editors, *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pages 67–83. PMLR, 09–12 Jul 2020. URL https://proceedings.mlr.press/v125/agarwal20b.html.

[5] Jordan T. Ash, Cyril Zhang, Surbhi Goel, Akshay Krishnamurthy, and Sham M. Kakade. Anti-concentrated confidence bonuses for scalable exploration. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=RXQ-FPbQYVn.

[6] Mohammad Gheshlaghi Azar, Rémi Munos, and Hilbert J Kappen. Minimax pac bounds on the sample complexity of reinforcement learning with a generative model. *Machine learning*, 91 (3):325–349, 2013.

[7] Philip Ball, Jack Parker-Holder, Aldo Pacchiano, Krzysztof Choromanski, and Stephen Roberts. Ready policy one: World building through active learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 591–601. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/ball20a.html.

[8] A. Bondeson and D.J. Ward. Stabilization of external modes in tokamaks by resistive walls and plasma rotation. *Physical Review Letters*, 72(17):2709–2712, 1994.

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

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[11] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1lJJnR5Ym.

[12] Kathryn Chaloner and Isabella Verdinelli. Bayesian experimental design: A review. *Statistical Science*, pages 273–304, 1995.

[13] Ian Char, Youngseog Chung, Willie Neiswanger, Kirthevasan Kandasamy, Andrew O Nelson, Mark Boyer, Egemen Kolemen, and Jeff Schneider. Offline contextual bayesian optimization. *Advances in Neural Information Processing Systems*, 32:4627–4638, 2019.

[14] Richard Y. Chen, Szymon Sidor, Pieter Abbeel, and John Schulman. UCB and infogain exploration via q-ensembles. *CoRR*, abs/1706.01502, 2017. URL http://arxiv.org/abs/1706.01502.

[15] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. 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. URL https://proceedings.neurips.cc/paper/2018/file/3de568f8597b94bda53149c7d7f5958c-Paper.pdf.

[16] Sebastian Curi, Felix Berkenkamp, and Andreas Krause. Efficient model-based reinforcement learning through optimistic policy search and planning. In *NeurIPS*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/a36b598abb934e4528412e5a2127b931-Abstract.html.

[17] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[18] Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian q-learning. In *Aaai/iaai*, pages 761–768, 1998.

[19] Richard Dearden, Nir Friedman, and David Andre. Model-based bayesian exploration. *CoRR*, abs/1301.6690, 1999. URL http://arxiv.org/abs/1301.6690.

[20] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897): 414–419, 2022.

[21] Marc Peter Deisenroth and Carl Edward Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 465–472, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

[22] Max E Fenstermacher, J Abbate, S Abe, T Abrams, M Adams, B Adamson, N Aiba, T Akiyama, P Aleynikov, E Allen, et al. Diii-d research advancing the physics basis for optimizing the tokamak approach to fusion energy. *Nuclear Fusion*, 62(4):042024, 2022.

[23] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine*

Learning Research, pages 1587–1596. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/fujimoto18a.html.

[24] Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *CoRR*, abs/1609.04436, 2016. URL http://arxiv.org/abs/1609.04436.

[25] R. J. Groebner, K. H. Burrell, and R. P. Seraydarian. Role of edge electric field and poloidal rotation in the l-h transition. *Physical Review Letters*, 64(25):3015–3018, 1990. ISSN 00319007. doi: 10.1103/PhysRevLett.64.3015.

[26] Arthur Guez, David Silver, and Peter Dayan. Efficient bayes-adaptive reinforcement learning using sample-based search. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1025–1033, Red Hook, NY, USA, 2012. Curran Associates Inc.

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

[28] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

[29] Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(6), 2012.

[30] José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. *Advances in neural information processing systems*, 27, 2014.

[31] David Humphreys, G Ambrosino, Peter de Vries, Federico Felici, Sun H Kim, Gary Jackson, A Kallenbach, Egemen Kolemen, J Lister, D Moreau, et al. Novel aspects of plasma control in iter. *Physics of Plasmas*, 22(2):021806, 2015.

[32] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

[33] Sham Machandranath Kakade. *On the sample complexity of reinforcement learning*. University of London, University College London (United Kingdom), 2003.

[34] Sanket Kamthe and Marc Deisenroth. Data-efficient reinforcement learning with probabilistic model predictive control. In Amos Storkey and Fernando Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 1701–1710. PMLR, 09–11 Apr 2018. URL https://proceedings.mlr.press/v84/kamthe18a.html.

[35] J Zico Kolter and Andrew Y Ng. Near-bayesian exploration in polynomial time. In *Proceedings of the 26th annual international conference on machine learning*, pages 513–520, 2009.

[36] Kimin Lee, Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. In *International Conference on Machine Learning*, pages 6131–6141. PMLR, 2021.

[37] 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. *arXiv preprint arXiv:1509.02971*, 2015.

[38] David JC MacKay et al. Bayesian nonlinear modeling for the prediction competition. *ASHRAE transactions*, 100(2):1053–1062, 1994.

[39] Viraj Mehta, Ian Char, Willie Neiswanger, Youngseog Chung, Andrew Nelson, Mark Boyer, Egemen Kolemen, and Jeff Schneider. Neural dynamical systems: Balancing structure and flexibility in physical prediction. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 3735–3742. IEEE, 2021.

[40] Viraj Mehta, Biswajit Paria, Jeff Schneider, Stefano Ermon, and Willie Neiswanger. An experimental design perspective on model-based reinforcement learning. In *International Conference on Learning Representations*, 2022.

[41] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[42] Radford M Neal. *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto, 1995.

[43] Willie Neiswanger, Ke Alexander Wang, and Stefano Ermon. Bayesian algorithm execution: Estimating computable properties of black-box functions using mutual information. In *International Conference on Machine Learning*. PMLR, 2021.

[44] Nikolay Nikolov, Johannes Kirschner, Felix Berkenkamp, and Andreas Krause. Information-directed exploration for deep reinforcement learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Byx83s09Km.

[45] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/8d8818c8e140c64c743113f563cf750f-Paper.pdf.

[46] Ian Osband, Benjamin Van Roy, Daniel J. Russo, and Zheng Wen. Deep exploration via randomized value functions. *Journal of Machine Learning Research*, 20(124):1–62, 2019. URL http://jmlr.org/papers/v20/18-339.html.

[47] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[48] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5062–5071. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/pathak19a.html.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[50] Luis Pineda, Brandon Amos, Amy Zhang, Nathan O. Lambert, and Roberto Calandra. Mbrl-lib: A modular library for model-based reinforcement learning. *Arxiv*, 2021. URL https://arxiv.org/abs/2104.10159.

[51] Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolinek, and Georg Martius. Sample-efficient cross-entropy method for real-time planning. *arXiv preprint arXiv:2008.06389*, 2020.

[52] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[53] Stephane Ross, Brahim Chaib-draa, and Joelle Pineau. Bayes-adaptive pomdps. In *NIPS*, pages 1225–1232, 2007.

[54] Daniel Russo and Benjamin Van Roy. Learning to optimize via information-directed sampling. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/301ad0e3bd5cb1627a2044908a42fdc2-Paper.pdf.

[55] Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen, et al. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1):1–96, 2018.

[56] Ilya O Ryzhov and Warren B Powell. Information collection on a graph. *Operations Research*, 59(1):188–201, 2011.

[57] Ilya O Ryzhov, Martijn RK Mes, Warren B Powell, and Gerald van den Berg. Bayesian exploration for approximate dynamic programming. *Operations research*, 67(1):198–214, 2019.

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

[59] Matthias Schultheis, Boris Belousov, Hany Abdulsamad, and Jan Peters. Receding horizon curiosity. In *Conference on robot learning*, pages 1278–1288. PMLR, 2020.

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

[61] Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5779–5788. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/shyam19a.html.

[62] JA Stillerman, TW Fredian, KA Klare, and G Manduchi. Mdsplus data acquisition system. *Review of Scientific Instruments*, 68(1):939–942, 1997.

[63] Malcolm Strens. A bayesian framework for reinforcement learning. In *ICML*, volume 2000, pages 943–950, 2000.

[64] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998. ISBN 0-262-19398-1. URL http://www.cs.ualberta.ca/%7Esutton/book/ebook/the-book.html.

[65] Matthew Tesch, Jeff Schneider, and Howie Choset. Expensive function optimization with stochastic binary outcomes. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1283–1291, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL https://proceedings.mlr.press/v28/tesch13.html.

[66] Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. End-to-end model-free reinforcement learning for urban driving using implicit affordances. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7153–7162, 2020.

[67] Alexander Tschantz, Beren Millidge, Anil K Seth, and Christopher L Buckley. Reinforcement learning through active inference. *arXiv preprint arXiv:2002.12636*, 2020.

[68] Christopher KI Williams and Carl Edward Rasmussen. Gaussian processes for regression. 1996.

[69] James Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc Deisenroth. Efficiently sampling functions from gaussian process posteriors. In *International Conference on Machine Learning*, pages 10292–10302. PMLR, 2020.

[70] Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N Zare, and Patrick Riley. Optimization of molecules via deep reinforcement learning. *Scientific reports*, 9(1):1–10, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] Assumptions were fully stated for all derivations.
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] All code and instructions are included in supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] All training details are specified in the paper or in the code.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We show error bars computed via the standard error of the mean performance between runs.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We detail computational cost (see Sec A.2).
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] All existing assets are cited.
   (b) Did you mention the license of the assets? [Yes] All licenses are properly attributed.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] New assets are included in supplementary material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] No human data was used.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] No data with personally identifiable information was used.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Implementation Details

A.1 Derivation of Computational Cost

In this section, we derive the computational complexity of the TIP algorithm. For simplicity, we focus on a single TIP planning iteration as might be done at each replanning in closed-loop control or at the start of a trial in open-loop control. In order to keep the analysis general, we assume that the chosen planning algorithm requires p accesses to the model where h actions are sequentially executed, giving ph total queries per planner execution. We also assume that the numbers of inducing points and basis functions used in our GP posterior function sampling are constant as are the Monte Carlo hyperparameters m, n.

The TIP algorithm consists of the following major operations:

- Sample T_i' for $j \in [m]$: $O(nmN)$ total cost from sampling algorithm, where N is the dataset size.
- Sample τ_{ij}^* for $i \in [n], j \in [m]$: $phHnm$ total cost from running the planner (ph posterior function queries) H times for each sampled τ^*, where H is the MDP horizon.
- Compute Cholesky decomposition for each $D \cup \tau_{ij}^*$. This takes a total of $O(nm(N+H)^3)$ operations as the augmented dataset is of size $N+H$ and Cholesky decompositions are $O(d^3)$ in the matrix size d.
- Compute posterior covariance $\Sigma^{S'} \mid D \cup \tau_{ij}$ for all τ_{ij}. This involves several matrix operations but the most computationally intensive is solving h triangular systems of size $(N+H) \times (N+H)$, which each take $O((N+H)^2)$ time. So the total computation here is $O(pnmh(N+H)^2)$.
- Compute determinants of covariance matrices for each of p queries and nm augmented datasets $D \cup \tau_{ij}$. Each of these operations is over a matrix of size $h \times h$ and therefore costs $O(h^3)$. So the total cost is $O(pnmh^3)$.

Summing these costs gives $O(nmN + phHnm + nm(N+H)^3 + pnmh(N+H)^2 + pnmh^3)$. Clearly the third term dominates the first, the fourth dominates the second, and since $H > h$, the fourth dominates the fifth. So, the computational cost can be summarized as $O\left(nm\left((N+H)^3 + ph(N+H)^2\right)\right)$.

A.2 Wall Times

Though TIP and oTIP are designed for applications where samples are expensive and computation is relatively inexpensive, we present in this section data on the running time of these methods. We ran all experiments on a shared research cluster available to us on large machines with hundreds of GB of memory and between 24 and 88 CPU cores. In general our implementation did not make use of more than 20 CPU cores concurrently. In Table 3, we give the running time of the phases of the TIP algorithm. We note that the bulk of the computation in the planning procedure actually goes towards the just-in-time compilation of the JAX code that computes the cost function C_{τ^*} on sampled future trajectories. In order to allow for this compilation cost, we modified the iCEM algorithm from [51] to take fixed batch sizes as the compilation (e.g. for the β tracking problem) takes approximately 90% of the time required for planning. Unfortunately this compilation process must be repeated at every iteration due to the limitations of the JAX compiler. We believe that a similarly JIT-compiled implementation of the planning algorithm for sampling τ^* on posterior samples could lead to a substantial speedup and a more flexible compiler could do more still.

A.3 GP Model Details

For all of our experiments, we use a squared exponential kernel with automatic relevance determination [38, 42]. The parameters of the kernel were estimated by maximizing the likelihood of the parameters after marginalizing over the posterior GP [68].

To optimize the transition function, we simply sampled a set of points from the domain, evaluated the acquisition function, and chose the maximum of the set. This set was chosed uniformly for every problem but β + Rotation and Reacher, for which we chose a random subset of $\cup_i \cup_j \tau_{ij}^*$ (the posterior

Control Problem	Pendulum	Cartpole	β Tracking	β + Rotation	Reacher
Sample τ^* mn times	24	31	7	25	130
Plan actions that minimize C_{τ^*}	16	15	15	50	295
Total for TIP Iteration	40	46	22	75	425
Evaluation for one episode	5-20	2-10	2-5	3 - 18	100-500

Table 3: Runtime in seconds for the phases of the TIP algorithm on all problems when run on the authors' CPU machines. The ranges given show the runtime for the operation at the beginning and at the end of training, as some operations run longer as more data is added.

Control Problem	Pendulum	Cartpole	β Tracking	β + Rotation	Reacher
Number of samples	25	30	25	50	100
Number of elites	3	6	3	8	15
Planning horizon	20	15	5	5	15
Number of iCEM iterations	3	5	3	5	5
Replanning Period	6	1	2	1	1

Table 4: Hyperparameters used for optimization in MPC procedure for closed-loop control problems.

samples of the optimal trajectory) since the space of samples is 10-dimensional and uniform random sampling will not get good coverage of interesting regions of the state space.

A.4 Cost Function Details

We set $n = 15$ and $m = 1$ for our Monte Carlo estimate of the cost function for each problem.

A.5 Details on Planning Method

As mentioned in the main text, we use the iCEM method from Pinneri et al. [51] with one major modification: a fixed sample batch size. This is in order to take advantage of the JIT compilation features of JAX and avoid recompiling code for each new batch size.

In Tables 4 and 5, we present the hyperparameters used for the planning algorithm across each problem. The same hyperparameters were used for the TIP, MPC, EIG_T, DIP, sDIP, and sTIP methods. As recommended by the original paper, we use $\beta = 3$ for the scaling exponent of the power spectrum density of sampled noise for action sequences, $\gamma = 1.25$ for the exponential decay of population size, and $\xi = 0.3$ for the amount of caching.

B Description of Comparison Methods

We compare against 14 different methods across open and closed-loop problems. Of these, 7 used the same model and planning algorithm (including hyperparameters) as TIP and oTIP. **DIP** and **oDIP** use the cost function $C(\tau) = -\mathbb{H}\left[T(S') \mid D\right]$ and **sDIP** (summed DIP) uses the cost function $C(\tau) = -\sum_{i=0}^{h} \mathbb{H}\left[T(s_i, a_i) \mid D\right]$. These are all pure exploration methods, but DIP and oDIP are more sophisticated in that they plan for future observations with a large amount of *joint* information

Control Problem	Nonlinear Gain 1	Nonlinear Gain 2	Lava Path
Number of samples	50	50	25
Number of elites	6	6	4
Planning horizon	10	10	20
Number of iCEM iterations	6	8	6

Table 5: Hyperparameters used for optimization in MPC procedure for open-loop control problems.

as opposed to sTIP which sums the individual information expected at each timestep. oDIP is simply the open loop variant of DIP. **EIG**$_T$ uses the same objective as sDIP but operates in the TQRL setting, querying points that approximately maximize the predictive entropy of the dynamics model. **BARL** similarly operates in the TQRL setting but uses the EIG_{τ^*} acquisition function from Mehta et al. [40]. We use the authors' implementation of that work for comparison. **MPC** uses C_g from (4) and plans to directly maximize expected rewards. This method can be seen as quite similar to Kamthe and Deisenroth [34] and a close cousin of Deisenroth and Rasmussen [21] in that it optimizes the same objective with a similar model. **oMPC** is simply the open loop variant of MPC.

Besides these methods which directly compare cost functions, we include 8 additional baselines from published work. **PETS** is a method given in Chua et al. [15] which uses a similar cross-entropy based planner and a probabilistic ensemble of neural networks for an uncertainty-aware estimate of the dynamics. PETS also plans to minimize C_g. **HUCRL** [16] learns a policy via backpropagation through time using a hallucinated perturbation to the dynamics that maximizes discounted rewards subject to the one-step confidence interval of the dynamics. HUCRL also uses a probabilistic ensemble of neural networks. Using the same implementation we also tested Thompson Sampling (**TS**), which acts optimally according to a network drawn from the posterior over models, and **BPTT** which plans to minimize C_g using a neural network policy and backpropagation through time. BPTT can also be viewed as a cousin of PILCO [21] as it attempts to take stochastic gradients of the expected cost. We also compare against **SAC** [27], **TD3** [23], and **PPO** [58]. SAC uses entropy bonuses to approximate Boltzmann exploration in an actor-critic framework. TD3 and PPO include various tricks for stable learning and add Ornstein-Uhlenbeck noise in order to explore.

For our FEEF implementation, we took hyperparameters from the most similar comparison environments in that paper and used them for our results. We tried several values for 'expl_weight' in order to se whether we were inadequately balancing exploration and exploitation. Ultimately we saw an 'expl_weight' of 0.1 was the best value.

We used the author's implementation of RHC. RHC makes strong assumptions on the form of the reward function by assuming that all problems are regulation problems where the goal is to drive the system to a given state and keep it there (with some cost for actuation). We were able to pass the targets for all of our problems (which may change between episodes) to the RHC controller. We did a light hyperparameter search tuning the number of random Fourier features used in the Bayesian linear model in this method. Ultimately we were disappointed in the performance of RHC when applied to our problems. We believe that this might be due to its undirected uncertainty sampling objective and relatively constrained model of environment dynamics.

C Description of Control Problems

C.1 Plasma Control Problems

The plasma control problems are based on controlling a tokamak, a toroidally shaped device for confining a thermonuclear plasma using magnetic fields. Achieving net positive energy from fusion requires confining a plasma at high enough temperature and density long enough for hydrogen isotopes to collide and fuse. However, as the temperature and density are increased, a wide variety of instabilities can occur which degrade confinement, leading to a loss of energy. Full physics simulation of tokamak plasmas requires 10s-1000s of CPU hours to simulate a single trajectory, and often require hand tuning of different parameters to achieve accurate results. Following the work of Abbate et al. [2], each of our plasma control problems used neural networks trained on data as the ground truth dynamics models. We used the MDSPlus tool [62] to fetch historical discharges from the DIII-D tokamak in San Diego [22]. In total, we trained our models on 1,479 historical discharges. The data was pre-processed following the procedure outlined in Abbate et al. [2]. We describe how each environment was constructed in more detail below.

β **Tracking** In this environment the goal is to adjust the total injected power (PINJ) of the neutral beams so that the normalized plasma pressure, β_N (defined as the ratio of thermal energy in the plasma to energy in the confining magnetic fields), reaches a target value of 2%. Reliably controlling plasmas to sustain high performance is a major goal of research efforts for fusion energy, so even this simple scenario is of interest. The ground-truth dynamics model takes in the current β_N and PINJ, the β_N and PINJ at some Δt time in the past, and the PINJ at some Δt time in the future (we assume

that we have complete control over the values of PINJ at all times). Given these inputs, the model was trained to output what β_N will be Δt time into the future. In total, the state space is 4D and the action space is 1D. For this environment, we set $\Delta t = 200ms$, and we specify the reward function to be the negative absolute difference between the next β_N and the target $\beta_N = 2\%$.

β + Rotation Tracking This environment is a more complicated version of the β tracking environment in several ways. First of all, the controller now must simultaneously track both β_N and the core toroidal rotation of the plasma. To do so, the controller is also allowed to set the total torque injected (TINJ) of the neutral beams (DIII-D has eight neutral beam injectors at different positions around the tokamak, so it is generally possible to control both total power and total torque independently). Controlling both of these quantities simultaneously is of interest since rotation shear often results in better confinement and less chance of instabilities in the plasma [8, 25]. In addition, we assume a multi-task setting where the requested targets for β_N and rotation can be set every trajectory. Specifically, the β_N target is drawn from $U(1.5\%, 2.5\%)$ and the rotation target is drawn from $U(25, 125)$ krad/s every trajectory. These targets are appended to the state space.

The learned, ground-truth dynamics model is also more sophisticated here. In addition to the inputs and outputs used by the β tracking environment model, the inputs for this model also include rotation and TINJ at times t, $t - \Delta t$, and $t + \Delta t$ for TINJ only. This model receives additional information about the plasma (e.g. the shape of the plasma); however, we have assumed these inputs are fixed to reasonable values in order to avoid partial observability problems. In total, the state space of this problem is 10D (targets plus current and past observations for β_N, rotation, PINJ, and TINJ) and the action space is 2D (next PINJ and TINJ settings).

C.2 Robotics Problems

Pendulum The pendulum swing-up problem is the standard one found in the OpenAI gym [10]. The state space contains the angle of the pendulum and its first derivative and action space simply the scalar torque applied by the motor on the pendulum. The challenge in this problem is that the motor doesn't have enough torque to simply rotate the pendulum up from all positions and often requires a back-and-forth swing to achieve a vertically balanced position. The reward function here penalizes deviation from an upright pole and squared torque.

Cartpole The cartpole swing-up problem has 4-dimensional state (position of the cart and its velocity, angle of the pole and its angular velocity) and a 1-dimensional action (horizontal force applied to the cart). Here, the difficulty lies in translating the horizontal motion of the cart into effective torque on the pole. The reward function is a negative sigmoid function penalizing the distance betweent the tip of the pole and a centered upright goal position.

Reacher The reacher problem simulates a 2-DOF robot arm aiming to move the end effector to a randomly resampled target provided. The problem requires joint angles and velocities as well as an indication of the direction of the goal, giving an 8-dimensional state space along with the 2-dimensional control space.

D Additional Results

Due to space constraints in the main paper, we omitted results for the methods sDIP and BPTT. The are included alongside the rest in Table 6. They are outperformed across the board by TIP.

E Additional Related Work

E.1 Bayesian Exploration Techniques

Given unlimited computation and an accurate prior, solving the Bayes-adaptive MDP [53] gives an optimal tradeoff between exploration and exploitation by explicitly accounting for the updated beliefs that would result from future observations and planning to find actions that result in high rewards as quickly as can be managed given the current posterior. However, this is computationally expensive even in small finite MDPs and totally intractable in continuous settings. Kolter and Ng

Environment	TIP	sTIP	DIP	sDIP	MPC	PETS	SAC	TD3	PPO	FEEF	HUCRL	TS	BPTT	BARL	EIG_T
Pendulum	**21**	36	36	46	46	5.6k	7k	26k	14k	800	>50k	>50k	>50k	**21**	56
Cartpole	131	141	161	141	201	1.63k	32k	18k	>1M	>2.5k	>6k	>6k	>6k	**111**	121
β Tracking	**46**	76	276	131	76	330	12k	17k	39k	300	480	420	450	186	>1k
β + Rotation	**201**	>500	>500	>500	>500	400	30k	>50k	>50k	>2k	>5k	>5k	>5k	>500	>1k
Reacher	**251**	>400	>1k	>1k	751	700	23k	13k	>100k	>5k	6.6k	4.5k	3.7k	**251**	>1.5k

Table 6: **Sample Complexity Comparison of All Methods:** Median number of samples across 5 seeds required to reach 'solved' performance, averaged across 5 trials. We determine 'solved' performance by running an MPC policy (similar to the one used for evaluation) on the ground truth dynamics to predict actions. We record $> n$ when the median run is unable to solve the problem by the end of training after collecting n datapoints. The methods in the rightmost section operate in the TQRL setting and therefore have more flexible access to the MDP dynamics for data collection.

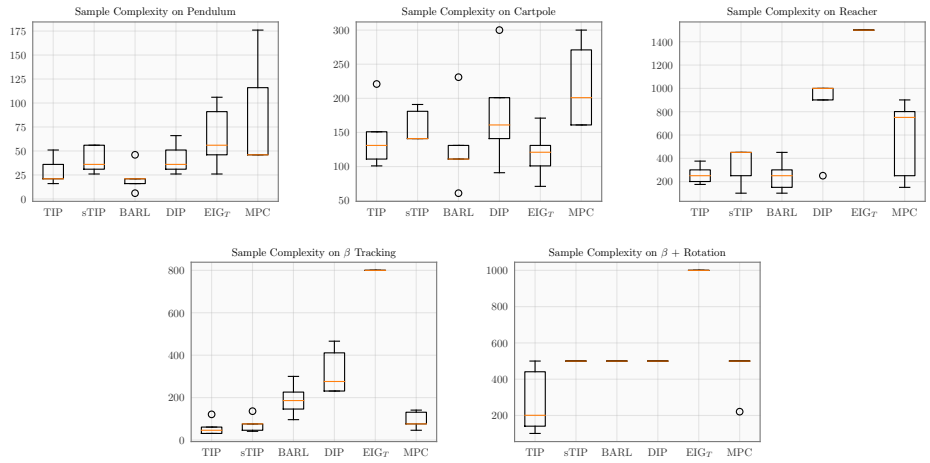

Figure 4: Box plots showing sample complexity figures across the 5 random seeds run. Each of these show for a given training run how many samples were needed to achieve the performance of an MPC controller given ground truth dynamics averaged across test episodes. We imputed the maximum number of samples for agents that failed to ever solve the problem on a given run.

[35] and Guez et al. [26] show that even approximating these techniques can result in substantial theoretical reductions in sample complexity compared to frequentist PAC-MDP bounds as in Kakade [33]. Another line of work [18, 19] uses the myopic value of perfect information as a heuristic for similar Bayesian exploration in the tabular MDP setting. Further techniques for exploration include knowledge gradient policies [57, 56], which approximate the value function of the Bayes-adaptive MDP and information-directed sampling (IDS) [54], which takes actions based on minimizing the ratio between squared regret and information gain over dynamics. This was extended to continuous-state finite-action settings using neural networks in Nikolov et al. [44]. Another very relevant recent paper [7] gives an acquisition strategy in policy space that iteratively trains a data-collection policy in the model that trades off exploration against exploitation using methods from active learning. Achterhold and Stueckler [3] use techniques from BOED to efficiently calibrate a Neural Process representation of a distribution of dynamics to a particular instance, but this calibration doesn't include information about the task. A tutorial on Bayesian RL methods can be found in Ghavamzadeh et al. [24] for further reference.

E.2 Gaussian Processes (GPs) in Reinforcement Learning

There has been substantial prior work using GPs [52] in reinforcement learning. Most well-known is PILCO [21], which computes approximate analytic gradients of policy parameters through the GP dynamics model while accounting for uncertainty. The original work is able to propagate the first 2 moments of the occupancy distribution through time using the GP dynamics and backpropagate gradients of the rewards to policy parameters. In [69], a method is developed for efficiently sampling functions from a GP posterior with high accuracy. One application show in their work is a method of using these samples to backpropagate gradients of rewards through time to policy paramters,

which can be interpreted as a different sort of PILCO implementation. Most related to our eventual MPC-based method is [34], which gives a principled probabilistic model-predictive control algorithm for GPs. We combine ideas from this paper, PETS [15], and the ability to sample posterior functions discussed above to give our eventual MPC component as discussed in Section 4.1.