Predictable MDP Abstraction for Unsupervised Model-Based RL

Seohong Park [1] **Sergey Levine** [1]

Abstract

A key component of model-based reinforcement learning (RL) is a dynamics model that predicts the outcomes of actions. Errors in this predictive model can degrade the performance of model-based controllers, and complex Markov decision processes (MDPs) can present exceptionally difficult prediction problems. To mitigate this issue, we propose **predictable MDP abstraction** (**PMA**): instead of training a predictive model on the original MDP, we train a model on a transformed MDP with a learned action space that only permits predictable, easy-to-model actions, while covering the original state-action space as much as possible. As a result, model learning becomes easier and more accurate, which allows robust, stable model-based planning or model-based RL. This transformation is learned in an *unsupervised* manner, before any task is specified by the user. Downstream tasks can then be solved with model-based control in a *zero-shot* fashion, without additional environment interactions. We theoretically analyze PMA and empirically demonstrate that PMA leads to significant improvements over prior unsupervised model-based RL approaches in a range of benchmark environments. Our code and videos are available at https://seohong.me/projects/pma/

1. Introduction

The basic building block of model-based reinforcement learning (RL) algorithms is a predictive model $\hat{p}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$, typically one that predicts the next state conditioned on the previous state and action in the given Markov decision process (MDP). By employing predictive models with planning or RL, previous model-based approaches have been shown to be effective in solving a variety of complex problems, ranging from robotics (Wu et al., 2022) to games (Schrittwieser et al., 2020), in a sample-efficient manner.

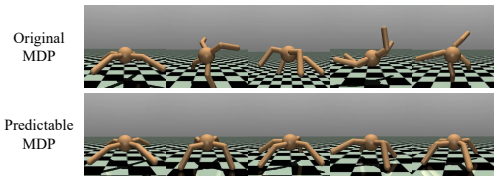


Figure 1. In the original Ant environment, some actions lead to unpredictable behaviors that are difficult to accurately model[2], which makes the learned dynamics model susceptible to catastrophic model exploitation. In our transformed predictable MDP, every transition is easy to model and predictable, which enables robust, stable model-based learning.

However, even small errors in a predictive model can cause a model-based RL algorithm to underperform, sometimes catastrophically (Hasselt et al., 2019; Jafferjee et al., 2020). This phenomenon, referred to as *model exploitation*, happens when a controller or policy exploits these errors, picking actions that the model erroneously predicts should lead to good outcomes. This issue is further exacerbated with long-horizon model rollouts, which accumulate prediction errors over time, or in complex MDPs, where accurately modeling all transitions is challenging. Previous approaches often try to address model exploitation by estimating model uncertainty (Chua et al., 2018) or only using the model for short rollouts (Buckman et al., 2018; Janner et al., 2019).

We take a different perspective on model-based RL to tackle this challenge: instead of training a predictive model on the original MDP, we apply model-based RL on top of an abstracted, simplified MDP. Namely, we first abstract the MDP into a simpler learned MDP with a transformed latent action space by *restricting* unpredictable actions, and then build the predictive model on this simplified MDP. Here, the transformed MDP is designed to be *predictable* in the sense that every transition in the new MDP is easy to model, while covering the original state-action space as much as possible. As a result, there is little room for catastrophic model exploitation compared to the original, possibly complex MDP, allowing robust model-based planning and RL. We illustrate an example of predictable MDP transformation in Figure 1.

We design a practical algorithm for learning predictable MDP abstractions in the setting of *unsupervised* model-

[1]University of California, Berkeley. Correspondence to: Seohong Park <seohong@berkeley.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[2]Even though the Ant environment is completely deterministic, it is very difficult to accurately model all the possible transitions due to its complex, contact-rich dynamics (see Section 7.3).

based RL, where the abstraction is learned in advance without any user-defined task or reward function. After unsupervised training, the learned MDP abstraction can be used to solve multiple different downstream tasks with a model-based controller in a *zero-shot* fashion, without any environment interactions or additional model training.

The desiderata of unsupervised predictable MDP abstraction are threefold. First, the latent actions in the transformed MDP should lead to predictable state transitions. Second, different latent actions should lead to different outcomes. Third, the transitions in the latent MDP should cover the original state-action space as much as possible. In this paper, we formulate these desiderata into an information-theoretic objective and propose a practical method to optimize it.

To summarize, our main contribution in this paper is to introduce a novel perspective on model-based RL by proposing **predictable MDP abstraction** (**PMA**) as an unsupervised model-based RL method, which abstracts the MDP by transforming the action space to minimize model errors. PMA can be combined with any existing model-based planning or RL method to solve downstream tasks in a zero-shot manner. We theoretically analyze PMA and discuss when our approach can be beneficial compared to classic model-based RL. Finally, we empirically confirm that PMA combined with model-based RL can robustly solve a variety of tasks in seven diverse robotics environments, significantly outperforming previous unsupervised model-based approaches.

2. Related Work

Model-based reinforcement learning. Model-based RL (MBRL) involves using a predictive model that estimates the outcomes of actions in a given environment. Previous model-based approaches utilize such learned models to maximize the reward via planning (Hernandaz & Arkun, 1990; Draeger et al., 1995; Deisenroth & Rasmussen, 2011; Lenz et al., 2015; Ebert et al., 2018; Chua et al., 2018; Hafner et al., 2019; Nagabandi et al., 2019), reinforcement learning (Heess et al., 2015; Feinberg et al., 2018; Buckman et al., 2018; Janner et al., 2019; Hafner et al., 2020; Nguyen et al., 2021), or both (Argenson & Dulac-Arnold, 2021; Sikchi et al., 2022; Hansen et al., 2022). Erroneous predictive models can yield deleterious effects on policy learning, which is known as the model exploitation problem (Ross & Bagnell, 2012; Janner et al., 2019; Kidambi et al., 2020; Kang et al., 2022). To avoid making suboptimal decisions based on incorrect models, prior works either restrict the horizon length of model rollouts (Janner et al., 2019) or employ various uncertainty estimation techniques, such as Gaussian processes (Rasmussen & Kuss, 2003; Deisenroth & Rasmussen, 2011) or model ensembles (Rajeswaran et al., 2017; Clavera et al., 2018; Kurutach et al., 2018; Chua et al., 2018; Nagabandi et al., 2019; Yu et al., 2020; Kidambi et al., 2020). Our work is orthogonal and complementary to these model-based RL algorithms. We propose an action representation learning method that abstracts the MDP into one that is *more predictable*, thus making model learning easier, which can be employed in combination with a variety of existing model-based RL methods.

Predictable behavior learning. Our main idea conceptually relates to prior work on predictable behavior learning. One such work is RPC (Eysenbach et al., 2021), which encourages the agent to produce predictable behaviors by minimizing model errors. SMiRL (Berseth et al., 2021) and IC2 (Rhinehart et al., 2021) actively seek stable behaviors by reducing uncertainty. While these methods incentivize the agent to behave in predictable ways or visit familiar states, they do not aim to provide a model-based RL method, instead utilizing the predictability bonus either for intrinsic motivation (Berseth et al., 2021; Rhinehart et al., 2021) or to improve robustness (Eysenbach et al., 2021). In contrast, we show that optimizing for predictability can lead to significantly more effective model-based RL performance.

MDP abstraction and hierarchical RL. MDP abstraction deals with the problem of building simplified MDPs that usually have simpler state or action spaces to make RL more tractable. State abstraction (Li et al., 2006; Watter et al., 2015; Ha & Schmidhuber, 2018; Gelada et al., 2019; Castro, 2019; Hafner et al., 2020) focuses on having a compact state representation to facilitate learning. Temporal abstraction and hierarchical RL (Sutton et al., 1999; Stolle & Precup, 2002; Bacon et al., 2017; Vezhnevets et al., 2017; Machado et al., 2017; Nachum et al., 2018; Eysenbach et al., 2019; Wulfmeier et al., 2021; Salter et al., 2022) aim to learn temporally extended behaviors to reduce high-level decision steps. Different from these previous approaches, we explore a lossy approach to MDP abstraction where the action space is transformed into one that only permits more predictable transitions, thus facilitating more effective model-based reinforcement learning.

Unsupervised reinforcement learning. The goal of unsupervised RL is to acquire primitives, models, or other objects that are useful for downstream tasks through unsupervised interaction with the environment. The process of learning a predictable MDP abstraction with our method corresponds to an unsupervised RL procedure. Prior unsupervised RL methods have used intrinsic rewards for maximizing state entropy (Lee et al., 2019; Yarats et al., 2021; Liu & Abbeel, 2021), detecting novel states (Pathak et al., 2017; Burda et al., 2019; Pathak et al., 2019), learning diverse goal-condition policies (Pong et al., 2020; Mendonca et al., 2021), or acquiring temporally extended skills (Gregor et al., 2016; Eysenbach et al., 2019; Sharma et al., 2020; Xie et al., 2020; Strouse et al., 2022; Park et al., 2022; Laskin et al., 2022). Notably, several unsupervised model-based approaches (Shyam et al., 2019; Sekar et al., 2020; Rajeswar et al., 2022) have shown that predictive models trained via

unsupervised exploration help solve downstream tasks efficiently. However, these methods only focus on finding novel transitions (*i.e.*, maximizing coverage), without considering their *predictability*. Maximizing coverage without accounting for predictability can lead to model errors, which in turn lead to model exploitation, as shown by Shyam et al. (2019) as well as in our experiments. Our method is also closely related to DADS (Sharma et al., 2020), an unsupervised skill discovery method that uses a similar mutual information (MI) objective to ours. However, the main focus of our work is different from DADS: while the goal of DADS is to acquire a set of temporally extended skills, analogously to other works on skill discovery, our focus is instead on transforming the action space into one that only permits predictable actions without temporal abstraction, maximally covering the transitions in the original MDP. We both theoretically and empirically show that this leads to significantly better performance in a variety of model-based RL frameworks.

3. Preliminaries and Problem Statement

We consider an MDP without a reward function, also referred to as a controlled Markov process (CMP), $\mathcal{M} := (\mathcal{S}, \mathcal{A}, \mu, p)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space, $\mu \in \mathcal{P}(\mathcal{S})$ denotes the initial state distribution, and $p : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ denotes the state transition distribution. We also consider a set of N downstream tasks $\mathcal{T} = \{T_0, T_1, \dots, T_{N-1}\}$, where each task corresponds to a reward function $r_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for $i \in [N]$. $[N]$ denotes the set of $\{0, 1, \dots, N-1\}$. We denote the supremum of the absolute rewards as $R = \sup_{\boldsymbol{s}\in\mathcal{S},\boldsymbol{a}\in\mathcal{A},i\in[N]} |r_i(\boldsymbol{s}, \boldsymbol{a})|$ and the discount factor as γ.

Problem statement. In this work, we tackle the problem of *unsupervised model-based RL*, which consists of two phases. In the first unsupervised training phase, we aim to build a predictive model in a given CMP without knowing the tasks. In the subsequent testing phase, we are given multiple task rewards in the same environment and aim to solve them only using the learned model without additional training; *i.e.*, in a zero-shot manner. Hence, the goal is to build a model that best captures the environment so that we can later robustly employ the model to solve diverse tasks.

4. Predictable MDP Abstraction (PMA)

Model-based RL methods typically learn a model $\hat{p}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$. However, naïvely modeling all possible transitions is error-prone in complex environments, and subsequent control methods (planning or policy optimization) can exploit these errors, leading to overoptimistic model-based estimates of policy returns and ultimately in poor performance. Previous works generally try to resolve this by restricting model usage (Buckman et al., 2018; Janner et al., 2019) or better estimating uncertainty (Chua et al., 2018).

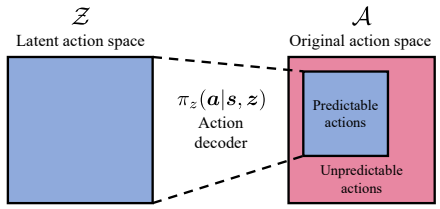


Figure 2. The action decoder reparameterizes the action space to only permit *predictable* transitions.

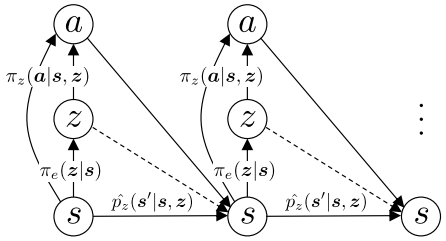


Figure 3. Architecture of PMA during unsupervised training. The exploration policy $\pi_e(\boldsymbol{z}|\boldsymbol{s})$ selects a latent action, which is decoded into the original action space by the action decoder $\pi_z(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$. The latent model $\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z})$ predicts outcomes in the latent MDP.

Different from previous approaches, our solution in this work is to transform the original MDP into a *predictable* latent MDP, in which every transition is predictable. Here, "predictable" also means that it is *easy to model*. Formally, we define the unpredictability of an MDP $\mathcal{M}$ as the minimum possible average model error ϵ with respect to a model class $\mathcal{F}$, a state-action distribution d, and a discrepancy measure D:

$$\epsilon = \inf_{f\in\mathcal{F}} \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim d(\boldsymbol{s},\boldsymbol{a})}[D(p(\cdot|\boldsymbol{s}, \boldsymbol{a})\|f(\cdot|\boldsymbol{s}, \boldsymbol{a}))]. \quad (1)$$

Intuitively, this measures the irreducible model error (*i.e.*, aleatoric uncertainty) of the environment given the capacity of the model class $\mathcal{F}$. We note that even in a completely deterministic environment, there may exist an irreducible model error if the model class $\mathcal{F}$ has a finite capacity (*e.g.*, $(512, 512)$-sized neural networks) and the environment dynamics are complex.

After transforming the original MDP into a simplified latent MDP, we solve downstream tasks on top of the latent MDP with model-based RL. Since the latent MDP is trained to be maximally predictable, there is little room for model exploitation compared to the original environment. We can thus later robustly employ the learned latent predictive model to solve downstream tasks with a model-based control method.

4.1. Architecture

The main idea of PMA is to *restrict* the original action space in a lossy manner so that it only permits predictable transitions. Formally, we transform the original MDP $\mathcal{M}$ into a predictable latent MDP defined as $\mathcal{M}_P := (\mathcal{S}, \mathcal{Z}, \mu, p_z)$

with the original state space $\mathcal{S}$, the latent action space $\mathcal{Z}$, and the latent transition dynamics $p_z : \mathcal{S} \times \mathcal{Z} \to \mathcal{P}(\mathcal{S})$.

PMA has three learnable components: an action decoder, a latent predictive model, and an exploration policy. The *action decoder* policy $\pi_z(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$ decodes latent actions into the original action space, effectively reparameterizing the action space in a lossy manner (Figure 2). The *latent predictive model* $\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z})$ predicts the next state in the latent MDP, which is *jointly* trained with the action decoder to make the learned MDP as predictable as possible. Finally, the *exploration policy* $\pi_e(\boldsymbol{z}|\boldsymbol{s})$ selects $\boldsymbol{z}$'s to train the PMA's components during the unsupervised training phase, maximizing the coverage in the original state space. We illustrate the components of PMA in Figure 3.

When building a latent predictive model $\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\theta})$, we derive our objective from a Bayesian perspective, integrating an information-theoretic representation learning goal with information gain on the posterior over the parameters $\boldsymbol{\theta}$. This naturally leads to the emergence of both an information-seeking exploration objective and an MI-based representation learning method for the latent action space.

After unsupervised training, once we get a reward function in the testing phase, we replace the exploration policy with a *task policy* $\pi(\boldsymbol{z}|\boldsymbol{s})$, which aims to select latent actions to solve the downstream task on top of our action decoder and predictive model. This task policy can either be derived based on a planner, or it could itself be learned via RL inside the learned model, as we will describe in Section 5. Both approaches operate in *zero shot*, in the sense that they do not require any additional environment interaction beyond the unsupervised phase.

4.2. Objective

We now state the three desiderata of our unsupervised predictable MDP abstraction: (*i*) The latent transitions $p_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z})$ in the predictable MDP should be as predictable as possible (*i.e.*, minimize aleatoric uncertainty). (*ii*) The outcomes of latent actions should be as different as possible from one another (*i.e.*, maximize action diversity to preserve as much of the expressivity of the original MDP as possible). (*iii*) The transitions in the predictable MDP should cover the original transitions as much as possible (*i.e.*, encourage exploration by minimizing epistemic uncertainty). These three goals can be summarized into the following concise information-theoretic objective:

$$\max_{\pi_z, \pi_e} I(\boldsymbol{S}'; (\boldsymbol{Z}, \boldsymbol{\Theta})|\boldsymbol{\mathcal{D}}), \tag{2}$$

where $\boldsymbol{\mathcal{D}}$ denotes the random variable (RV) of the entire training dataset up to and including the current state $\boldsymbol{S}$, $(\boldsymbol{S}, \boldsymbol{Z}, \boldsymbol{S}')$ denotes the RVs of $(\boldsymbol{s}, \boldsymbol{z}, \boldsymbol{s}')$ tuples from the policies, and $\boldsymbol{\Theta}$ denotes the RV of the parameters of the latent predictive model $\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\theta})$. Intuitively, this objective requires learning a latent action space, represented by π_z, as well as an exploration policy π_e in this latent action space, such that at each transition the resulting next state is easy to predict from the latent action and the model parameters. The inclusion of model parameters may seem like an unusual choice, but it leads naturally to an information gain exploration scheme that maximizes state coverage.

Equation (2) can be decomposed as follows, revealing three terms that directly map onto our desiderata:

$$\begin{aligned}
& I(\boldsymbol{S}'; (\boldsymbol{Z}, \boldsymbol{\Theta})|\boldsymbol{\mathcal{D}}) && (3)\\
=& I(\boldsymbol{S}'; \boldsymbol{Z}|\boldsymbol{\mathcal{D}}) + I(\boldsymbol{S}'; \boldsymbol{\Theta}|\boldsymbol{\mathcal{D}}, \boldsymbol{Z}) && (4)\\
=& I(\boldsymbol{S}'; \boldsymbol{Z}|\boldsymbol{S}) + I(\boldsymbol{S}'; \boldsymbol{\Theta}|\boldsymbol{\mathcal{D}}, \boldsymbol{Z}) && (5)\\
=& \underbrace{-H(\boldsymbol{S}'|\boldsymbol{S}, \boldsymbol{Z})}_{(i)\text{ predictability}} + \underbrace{H(\boldsymbol{S}'|\boldsymbol{S})}_{(ii)\text{ diversity}} \\
& + \underbrace{H(\boldsymbol{\Theta}|\boldsymbol{\mathcal{D}}, \boldsymbol{Z}) - H(\boldsymbol{\Theta}|\boldsymbol{\mathcal{D}}, \boldsymbol{Z}, \boldsymbol{S}')}_{(iii)\text{ information gain}}. && (6)
\end{aligned}$$

The first term in Equation (6) maximizes predictability by reducing the entropy of the next state distribution $p_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z})$, making the latent MDP maximally predictable. The second term increases the entropy of the marginalized next state distribution, effectively making the resulting states from different z's different from one another. The third term minimizes epistemic uncertainty by maximizing *information gain*, the reduction in the uncertainty of the predictive model's parameters after knowing $\boldsymbol{S}'$.

With the objective in Equation (2) as an intrinsic reward, we can optimize both the action decoder policy and exploration policy with RL. As a result, they will learn to produce the optimal $\boldsymbol{z}$'s and $\boldsymbol{a}$'s that in the long-term lead to maximal coverage of the original state space, while making the resulting latent MDP as predictable as possible. Also, we simultaneously train the latent predictive model $\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\theta})$ so that we can later use it for planning in the latent MDP.

4.3. Practical Implementation

We now describe a practical method to optimize our main objective in Equation (2). Since it is generally intractable to exactly estimate mutual information or information gain, we make use of several approximations.

Estimating $I(\boldsymbol{S}'; \boldsymbol{Z}|\boldsymbol{S})$. First, for the first two terms in Equation (6), we employ a variational lower-bound approximation as follows (Barber & Agakov, 2003):

$$\begin{aligned}
& I(\boldsymbol{S}'; \boldsymbol{Z}|\boldsymbol{S}) = -H(\boldsymbol{S}'|\boldsymbol{S}, \boldsymbol{Z}) + H(\boldsymbol{S}'|\boldsymbol{S}) && (7)\\
& \geq \mathbb{E}[\log \hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\phi}) - \log p_z(\boldsymbol{s}'|\boldsymbol{s})] && (8)\\
& \approx \underbrace{\mathbb{E}[\log \hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\phi}) - \log \frac{1}{L} \sum_{i=1}^{L} \hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}_i; \boldsymbol{\phi})]}_{:= r_{\text{emp}}(\boldsymbol{s}, \boldsymbol{z}, \boldsymbol{s}')}, && (9)
\end{aligned}$$

where $\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\phi})$ is a variational lower-bound (VLB) of $p_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z})$. Also, we approximate the intractable marginal

entropy term $H(\boldsymbol{S}'|\boldsymbol{S})$ with L random samples of $\boldsymbol{z}$'s from the latent action space (Sharma et al., 2020). These approximations provide us with a tractable intrinsic reward that can be optimized with RL. Here, we note that the second term in Equation (9) is a biased estimator for $\log p_z(\boldsymbol{s}'|\boldsymbol{s})$, since it is estimating an expectation inside the $\log$ with samples, but we found this approach to still work well in practice, and indeed multiple prior works have also explored such a biased estimator for mutual information objectives in RL (Sharma et al., 2020; Kim et al., 2021). Exploring unbiased lower bounds (Poole et al., 2019) for this MI objective is an interesting direction for future work.

Estimating information gain. Next, we need to estimate the information gain term in Equation (6). This term could be approximated directly using prior methods that propose exploration via information gain, *e.g.*, using a variational approximation (Houthooft et al., 2016). In our implementation, however, we use a more heuristic approximation that we found to be simpler to implement based on ensemble disagreement, motivated by prior works (Shyam et al., 2019; Ball et al., 2020; Sekar et al., 2020; Strouse et al., 2022). Namely, we first approximate the model posterior with an ensemble of E predictive models, $\{\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{z};\boldsymbol{\theta}_i)\}_{i\in[E]}$ with $p(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{E}\sum_i \delta(\boldsymbol{\theta}-\boldsymbol{\theta}_i)$. Each component models the transitions as conditional Gaussian with the mean given by a neural network and a unit diagonal covariance, $\boldsymbol{s}' \sim \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{s},\boldsymbol{z};\boldsymbol{\theta}_i), I)$. We then use the variance of the ensemble means with a coefficient β,

$$\mathbb{E}[\underbrace{\beta \cdot \mathrm{Tr}[\mathbb{V}_i[\boldsymbol{\mu}(\boldsymbol{s},\boldsymbol{z};\boldsymbol{\theta}_i)]]}_{:=r_{\text{dis}}(\boldsymbol{s},\boldsymbol{z},\boldsymbol{s}')}], \tag{10}$$

as a simple (though crude) estimator for information gain $I(\boldsymbol{S}';\boldsymbol{\Theta}|\boldsymbol{\mathcal{D}},\boldsymbol{Z})$. We provide a detailed justification in Appendix C. Intuitively, Equation (10) encourages the agent to explore states that have not been previously visited, where the ensemble predictions do not agree with one another.

Training PMA. With these approximations, we use $r_{\text{emp}}(\boldsymbol{s},\boldsymbol{z},\boldsymbol{s}') + r_{\text{dis}}(\boldsymbol{s},\boldsymbol{z},\boldsymbol{s}')$ as an intrinsic reward for the action decoder $\pi_z(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{z})$. For the exploration policy $\pi_e(\boldsymbol{z}|\boldsymbol{s})$, if we assume the action decoder is optimal, we can use $H(\boldsymbol{Z}|\boldsymbol{S}) + r_{\text{dis}}(\boldsymbol{s},\boldsymbol{z},\boldsymbol{s}')$ as an intrinsic reward. This can be optimized with any *maximum entropy* RL method. However, in practice, we find that it is sufficient in our experiments to simply use a maximum entropy policy $\pi_e(\cdot|\boldsymbol{s}) = \mathrm{Unif}(\mathcal{Z})$ since our action decoder also maximizes $r_{\text{dis}}(\boldsymbol{s},\boldsymbol{z},\boldsymbol{s}')$. Finally, we fit our VLB predictive model $\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{z};\boldsymbol{\phi})$ and ensemble models $\{\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{z};\boldsymbol{\theta}_i)\}$ using the $(\boldsymbol{s},\boldsymbol{z},\boldsymbol{s}')$ tuples sampled from our policies. We describe the full training procedure of PMA in Appendix F and Algorithm 1.

4.4. Connections to Prior Work

PMA's objective is related to several prior works in unsupervised RL. For example, if we set $\beta = 0$ and $\pi_e(\boldsymbol{z}_t|\boldsymbol{s}_t) = \boldsymbol{z}_{t-1}$ for $t \geq 1$, we recover DADS (Sharma et al., 2020), a previous unsupervised skill discovery method. Also, if we set $\pi_z(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{z}) = \boldsymbol{z}$, PMA becomes similar to prior unsupervised model-based approaches using disagreement-based intrinsic rewards (Shyam et al., 2019; Sekar et al., 2020). However, these methods either do not aim to cover the state-action space or do not consider predictability, which makes them suboptimal or unstable. In Section 7, we empirically compare PMA with these prior approaches and demonstrate that our full objective makes a substantial improvement over them. Additionally, we theoretically compare PMA with DADS in Appendix E.

5. Model-Based Learning with PMA

After completing the unsupervised training of PMA, we can employ the learned latent predictive model to optimize a reward function with model-based planning or RL. In this section, we present several ways to utilize our predictable MDP to solve downstream tasks in a *zero-shot* manner.

5.1. Model-Based Learning with PMA

After training the model, PMA can be combined with any existing model-based planning or RL method. Specifically, we can apply any off-the-shelf model-based RL method on top of the latent action space $\mathcal{Z}$ and the learned latent dynamics model $\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{z})$ to maximize downstream task rewards, where we use the mean of the ensemble model outputs as the latent dynamics model. By planning over the latent action space, we can effectively prevent model exploitation as hard-to-predict actions are filtered out.

In our experiments, we study two possible instantiations of model-based learning: one based on model-predictive control, and one based on approximate dynamic programming (*i.e.*, actor-critic RL), where the learned model is used as a "simulator" without additional environment samples. Note that both variants solve the new task in "zero shot," in the sense that they do not require any additional collection of real transitions in the environment.

Model predictive path integral (MPPI). MPPI (Williams et al., 2016) is a sampling-based zeroth-order planning algorithm that optimizes a short-horizon sequence of (latent) actions at each time step, executes the first action in the sequence, and then replans. We refer to Appendix F.2 and Algorithm 2 for the full training procedure.

Model-based policy optimization (MBPO). MBPO (Janner et al., 2019) is a Dyna-style model-based RL algorithm that trains a model-free RL method on top of truncated model-based rollouts starting from intermediate environment states. In our zero-shot setting, we train the task pol-

icy only using *purely* model-based rollouts, whose starting states are sampled from the restored replay buffer from unsupervised training. We refer to Appendix F.3 and Algorithm 3 for the full training procedure.

5.2. Addressing Distributional Shift

Using a fixed, pre-trained model for model-based control is inherently vulnerable to *distributional shift* as the test-time controller might find some "adversarial" $\boldsymbol{z}$ values that make the agent state out of distribution. This issue applies to our zero-shot setting as well, even though every transition in our latent MDP is trained to be predictable. As such, we explicitly penalize the agent for visiting out-of-distribution states, following prior offline model-based RL methods (Yu et al., 2020; Kidambi et al., 2020), which also deals with the same issue. As we already have an ensemble of latent predictive models, we use the following maximum disagreement between ensemble models (Kidambi et al., 2020) as a penalty with a coefficient λ:

$$u(\boldsymbol{s}, \boldsymbol{z}) = -\lambda \cdot \max_{i,j \in [E]} \|\mu(\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\theta}_i) - \mu(\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\theta}_j)\|^2. \quad (11)$$

During task-specific planning or RL, we add this penalty to the task reward, similarly to MOPO (Yu et al., 2020).

6. Theoretical Results

Predictable MDP abstraction is a lossy procedure. In this section, we theoretically analyze the degree to which this lossiness influences the performance of a policy in the abstracted MDP, and provide practical insights on when PMA can be useful compared to classic model-based RL. All formal definitions and proofs can be found in Appendix D.

6.1. PMA Performance Bound

We first state the performance bound of PMA. Formally, for the original MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mu, p, r)$, we define the MDP with a *learned* dynamics model $\hat{p}$ as $\hat{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, \mu, \hat{p}, r)$, where we assume that $\hat{p}$ is trained on the dataset $\mathcal{D}$ collected by $\pi_{\mathcal{D}}$. For our predictable latent MDP $\mathcal{M}_P = (\mathcal{S}, \mathcal{Z}, \mu, p_z, r)$, we similarly define $\hat{\mathcal{M}}_P = (\mathcal{S}, \mathcal{Z}, \mu, \hat{p}_z, r)$, $\mathcal{D}_P$, and $\pi_{\mathcal{D}_P}$. For a policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ in the original MDP, we define its corresponding latent policy that best mimics the original one as $\pi_z^{\phi^*}(\boldsymbol{z}|\boldsymbol{s})$, and its induced next-state distribution as $p_z^{\phi^*}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ (please see Appendix D for the formal definitions). We now state our performance bound of PMA with a learned latent dynamics model as follows:

Theorem 6.1. *If the abstraction loss, the model error, and the policy difference are bounded as follows:*

$$\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a}) \sim d^{\pi}(\boldsymbol{s},\boldsymbol{a})}[D_{\mathrm{TV}}(p(\cdot|\boldsymbol{s}, \boldsymbol{a}) \| p_z^{\phi^*}(\cdot|\boldsymbol{s}, \boldsymbol{a}))] \leq \epsilon_a,$$
$$\mathbb{E}_{(\boldsymbol{s},\boldsymbol{z}) \sim d^{\pi_{\mathcal{D}_P}}(\boldsymbol{s},\boldsymbol{z})}[D_{\mathrm{TV}}(p_z(\cdot|\boldsymbol{s}, \boldsymbol{z}) \| \hat{p}_z(\cdot|\boldsymbol{s}, \boldsymbol{z}))] \leq \epsilon'_m,$$
$$\mathbb{E}_{\boldsymbol{s} \sim d^{\pi_{\mathcal{D}_P}}(\boldsymbol{s})}[D_{\mathrm{TV}}(\pi_z^{\phi^*}(\cdot|\boldsymbol{s}) \| \pi_{\mathcal{D}_P}(\cdot|\boldsymbol{s}))] \leq \epsilon'_\pi,$$

the performance difference of π between the original MDP and the predictable latent model-based MDP can be bounded as:

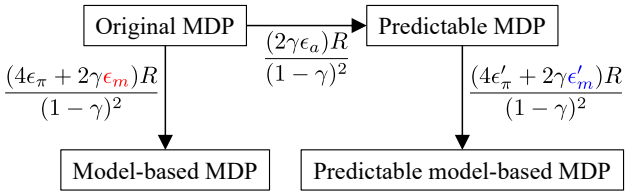


Figure 4. Performance bound between four MDPs. When $\epsilon_m \gg \epsilon_a + \epsilon'_m$, PMA provides a tighter bound than classic MBRL.

$$|J_{\mathcal{M}}(\pi) - J_{\hat{\mathcal{M}}_P}(\pi_z^{\phi^*})| \leq \frac{R}{(1-\gamma)^2}(2\gamma\epsilon_a + 4\epsilon'_\pi + 2\gamma\epsilon'_m). \quad (12)$$

Intuitively, PMA's performance bound consists of the following three factors: (*i*) the degree to which we lose from the lossy action decoder (ϵ_a), (*ii*) the model error in the latent predictive model (ϵ'_m), and (*iii*) the distributional shift between the data-collecting policy and the task policy (ϵ'_π). Hence, the bound becomes tighter if we have better state-action coverage and lower model errors, which is precisely what PMA aims to achieve (Equation (6)).

6.2. When Should We Use PMA over Classic MBRL?

To gain practical insights into PMA, we theoretically compare PMA with classic model-based RL. We first present the performance bound of classic MBRL (Janner et al., 2019):

Theorem 6.2. *If the model error and the policy difference are bounded as follows:*

$$\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a}) \sim d^{\pi_{\mathcal{D}}}(\boldsymbol{s},\boldsymbol{a})}[D_{\mathrm{TV}}(p(\cdot|\boldsymbol{s}, \boldsymbol{a}) \| \hat{p}(\cdot|\boldsymbol{s}, \boldsymbol{a}))] \leq \epsilon_m,$$
$$\mathbb{E}_{\boldsymbol{s} \sim d^{\pi_{\mathcal{D}}}(\boldsymbol{s})}[D_{\mathrm{TV}}(\pi(\cdot|\boldsymbol{s}) \| \pi_{\mathcal{D}}(\cdot|\boldsymbol{s}))] \leq \epsilon_\pi,$$

the performance difference of π between $\mathcal{M}$ and $\hat{\mathcal{M}}$ can be bounded as:

$$|J_{\mathcal{M}}(\pi) - J_{\hat{\mathcal{M}}}(\pi)| \leq \frac{R}{(1-\gamma)^2}(4\epsilon_\pi + 2\gamma\epsilon_m). \quad (13)$$

By comparing Equation (12) and Equation (13), we can see that PMA leads to a tighter bound when $4\epsilon_\pi + 2\gamma\epsilon_m > 2\gamma\epsilon_a + 4\epsilon'_\pi + 2\gamma\epsilon'_m$ (Figure 4). Intuitively, this condition corresponds to $\epsilon_m \gg \epsilon_a + \epsilon'_m$, if we assume that the data collection policies in both cases have a similar divergence from the desired policy π. This indicates that when the reduction in the model error by having predictable transitions outweighs the abstraction loss, PMA can be more beneficial than classic MBRL.

When can PMA be practically useful? PMA is useful when the optimal policies for the tasks mainly consist of predictable transitions so that we can reduce the model error

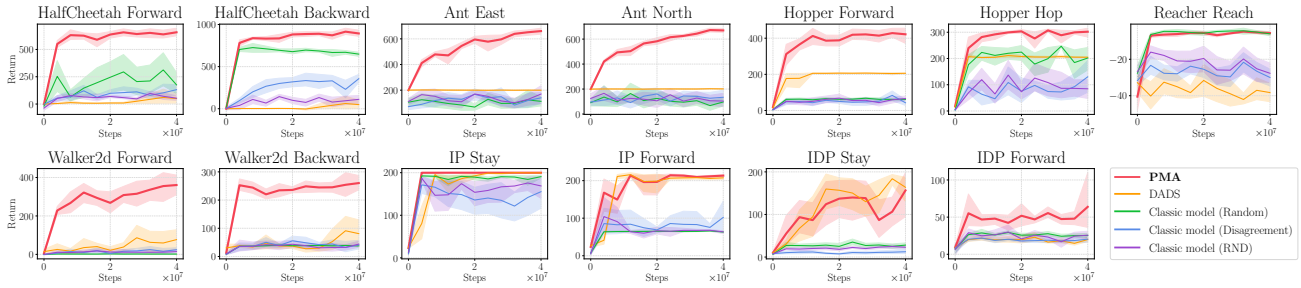


Figure 5. Comparison of periodic zero-shot planning (MPPI combined with the MOPO penalty) performances among unsupervised model-based RL methods. PMA demonstrates the best performance in most tasks, especially in Ant and Walker2d.

ϵ'_m while maintaining a small ϵ_a (the average abstraction loss over the transition distribution of the optimal policies *of our interest*). For instance, in real-world driving scenarios, we can achieve most of our driving purposes without modeling (and even by actively avoiding) unpredictable behaviors like breaking the car in diverse ways, which makes PMA beneficial. Similar arguments can be applied to many robotics environments, as we empirically demonstrate in Section 7. On the other hand, we could imagine MDPs where optimal behavior requires intentionally visiting unpredictable regions of the state space, in which case PMA could be suboptimal.

7. Experiments

In our experiments, we study the performance of PMA as an unsupervised model-learning algorithm for downstream *zero-shot* model-based RL, and analyze the degree to which PMA can learn more predictable models that enable longer-horizon simulated rollouts. In particular, we aim to answer the following questions: (*i*) Does PMA lead to better zero-shot task performances in diverse tasks? (*ii*) Does PMA enable robust longer-horizon planning, without suffering from model exploitation? (*iii*) Does PMA learn more predictable models?

Experimental setup. Since PMA does not require access to the task reward during the model training process, we focus our comparisons on other *unsupervised* model-based RL methods that operate under similar assumptions: a pre-training phase with interactive access to the MDP but not its reward function, followed by a *zero-shot* evaluation phase. Previous unsupervised model-based approaches (Shyam et al., 2019; Pathak et al., 2019; Sekar et al., 2020; Rajeswar et al., 2022) typically pre-train a classic dynamics model of the form $\hat{p}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ using data gathered by some exploration policy. We consider three of them as our baselines: classic models (CMs) $\hat{p}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ trained with (*i*) random actions ("Random"), (*ii*) ensemble disagreement-based exploration ("Disagreement") (Pathak et al., 2019), which was previously proposed as an unsupervised data collection scheme for model learning in several works (Shyam et al., 2019; Sekar et al., 2020; Rajeswar et al., 2022), and (*iii*) random network distillation ("RND") (Burda et al., 2019), another data collection method considered by Rajeswar et al. (2022). We also compare to DADS (Sharma et al., 2020), a previous unsupervised skill discovery method that also learns a latent action dynamics model $\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z})$ but aims to find compact, temporally extended behaviors, rather than converting the original MDP into a more predictable one. For the benchmark, we test PMA and the four previous methods on seven MuJoCo robotics environments (Todorov et al., 2012; Brockman et al., 2016) with 13 diverse tasks. We note that, in our experiments, we always use an ensemble disagreement penalty (MOPO penalty, Section 5.2) individually tuned for every method, task, and controller, to ensure fair comparisons. Every experiment is run with 8 random seeds and we present 95% confidence intervals in the plots.

7.1. Model-Based Planning with PMA

In order to examine whether PMA leads to better planning performance, we first evaluate the models learned by PMA and each of the prior approaches using zero-shot planning for a downstream task. PMA and DADS use latent models $\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z})$, while the other methods all use "classic" models of the form $\hat{p}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$, and differ only in their unsupervised data collection strategy. We perform the comparison on seven MuJoCo environments (HalfCheetah, Ant, Hopper, Walker2d, InvertedPendulum ("IP"), InvertedDoublePendulum ("IDP"), and Reacher) with 13 tasks. During unsupervised training of these methods, we periodically run MPPI planning (Section 5.1) combined with the MOPO penalty on the downstream tasks (these trials are not used for model training, which is completely unaware of the task reward), and report its results in Figure 5. The results show that PMA achieves the best performance in most tasks. Especially, PMA is the only successful unsupervised model-based method in Ant, whose complex, contact-rich dynamics make it difficult for classic models to succeed because erroneous model predictions often result in the agent flipping over. PMA successfully solves the tasks since our predictable abstraction effectively prevents such errors.

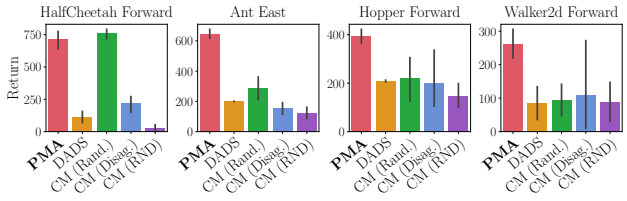


Figure 6. Comparison of unsupervised model-based RL methods using MBPO combined with the MOPO penalty. PMA mostly outperforms the prior approaches often by large margins.

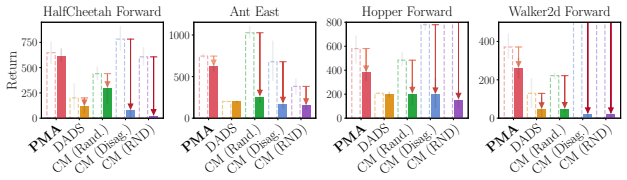
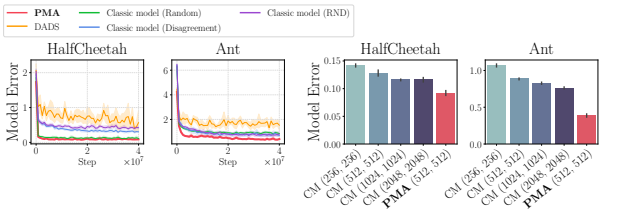


Figure 7. Performances of SAC combined with the MOPO penalty trained on purely model-based full-length rollouts. Dotted boxes indicate predicted returns and solid boxes indicate true returns. While classic models suffer from model exploitation in this long-horizon setting, as indicated by the drop from the predicted return to the actual return, PMA suffers a modest drop from the predicted return, and performs significantly better.

7.2. Model-Based RL with PMA

While the planning method used in the previous section uses the learned models with short-horizon MPC, we can better evaluate the capacity of the PMA model and the baselines to make faithful long-horizon predictions by using them with a long-horizon model-free RL procedure, essentially treating the model as a "simulator." We study two approaches in this section. The first is based on MBPO (Janner et al., 2019), which we describe in Section 5.1. The second approach is SAC (Haarnoja et al., 2018a) on top of full-length model-based rollouts, which in some sense is the most literal way to use the learned model as a "simulator" of the true environment. This second approach requires significantly longer model-based rollouts (up to 200 time steps), and though it performs worse in practice, it provides a much more stringent test of the models' ability to make long-horizon predictions without excessive error accumulation. In both evaluation schemes, we use the MOPO penalty to prevent distributional shifts.

Figure 6 and Figure 7 present the results with MBPO and SAC, respectively. In both settings, PMA mostly outperforms prior model-based approaches, suggesting that PMA is capable of making reliable long-horizon predictions. Also, by comparing the Hopper and Walker2d performances of Figure 5 and Figure 6, we find that classic models fail with MPPI and require a complex controller like MBPO to succeed, whereas PMA with a simple planner can achieve similar results to MBPO owing to its predictable dynamics. In the full-length SAC plots in Figure 7, we compare the models' predicted returns and the actual returns. We find that the drops from PMA's predicted returns to actual returns are generally modest, which indicates that our predictable abstraction effectively prevents model exploitation. DADS similarly shows small performance differences as it also restricts actions, but the absolute performance of DADS falls significantly behind PMA due to its limited coverage of the state-action space. On the other hand, classic models tend to be erroneously optimistic about the returns because of the complex dynamics, suffering from model exploitation.



(a) Model errors of various methods (b) Model errors of various network sizes

Figure 8. (a) PMA achieves the lowest model errors due to our predictability objective (videos). (b) Even in deterministic environments, it is challenging to completely reduce the errors in classic models even with larger neural networks.

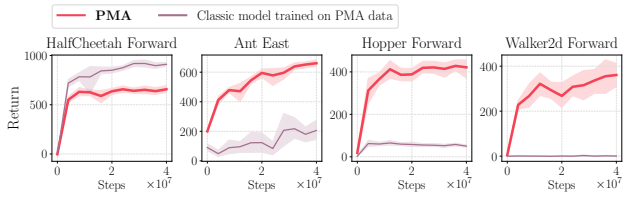


Figure 9. MPPI performance comparison between PMA and classic models trained with the data collected by PMA. While HalfCheetah does not benefit from action reparameterization, having a separate latent action space is crucial in the other environments.

7.3. Analysis

Can model errors be reduced by simply increasing the model size? We first compare the average mean squared errors of predicted (normalized) states of the five methods in HalfCheetah and Ant, and report the results in Figure 8a. In both environments, PMA exhibits the lowest model error, as it is trained to be maximally predictable. To examine whether this degree of error can be achieved by simply increasing the size of a classic model $\hat{p}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$, we train classic models with random actions using four different model sizes, ranging from two 256-sized hidden layers to two 2048-sized ones. Figure 8b shows the results, which suggest that there are virtually irreducible model errors even in deterministic environments due to their complex, contact-rich dynamics. On the other hand, PMA seeks to model not the entire MDP but only the "predictable" parts of the action space, which reduces model errors and thus makes it amenable to model-based learning.

Data restriction vs. action reparameterization. PMA serves both (*i*) data restriction by not selecting unpredictable actions and (*ii*) action reparameterization by having a separate latent action space. To dissect these two effects, we additionally consider a classic model $\hat{p}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ trained with the same data used to train PMA. We compare the periodic MPPI performances of this setting and PMA in Figure 9. The results suggest that while data restriction, in combination with the MOPO penalty, is sufficient in HalfCheetah, having a separate latent action space is crucial in the other "unstable" environments with early termination or irreversible states (*e.g.*, flipping over in Ant). This is because while the MOPO penalty at test time only prevents short-term deviations from the data distribution, PMA trained with *RL* considers long-term predictability, which effectively prevents selecting seemingly benign actions that could eventually cause the agent to lose balance (which corresponds to unpredictable behavior).

We refer to our project page and Appendix A for qualitative results and Appendix B for an ablation study.

8. Conclusion

We presented predictable MDP abstraction (PMA) as an unsupervised model-based method that builds a latent MDP by reparameterizing the action space to only allow predictable actions. We formulated its objective with information theory and theoretically analyzed the suboptimality induced by the lossy training scheme. Empirically, we confirmed that PMA enables robust model-based learning, exhibiting significant performance improvements over prior approaches.

Limitations. One limitation of PMA is that it is a lossy procedure. While we empirically demonstrated that its improved predictability outweighs the limitations imposed by the restriction of the action space in our experiments, PMA might be suboptimal in tasks that require unpredictable or highly complex behaviors, such those as discussed in Section 6.2, and in general it may be difficult to guarantee that the abstractions learned by PMA are good for every downstream task (though such guarantees are likely difficult to provide for any method). Also, PMA requires tuning the coefficient β to maintain a balance between predictability and the state-action space coverage. Nonetheless, we believe that methods that aim to specifically model the *predictable* parts of an MDP hold a lot of promise for future model-based RL methods. Future work could explore hybridizing such techniques with model-free approaches for handling the "unpredictable" parts, further study effective data collection strategies or methods that can utilize previously collected offline data, and examine how such approaches could be scaled up to more complex and high-dimensional observation spaces, where training directly for predictability could lead to even more significant gains in performance.

Acknowledgement

We would like to thank Benjamin Eysenbach for insightful discussions about the initial idea, Mitsuhiko Nakamoto, Jaekyeom Kim, and Youngwoon Lee for discussions about implementation and presentation, and RAIL members and anonymous reviewers for their helpful comments. Seohong Park was partly supported by Korea Foundation for Advanced Studies (KFAS). This research used the Savio computational cluster resource provided by the Berkeley Research Computing program at UC Berkeley and was partly supported by AFOSR FA9550-22-1-0273 and the Office of Naval Research.

References

Ajay, A., Kumar, A., Agrawal, P., Levine, S., and Nachum, O. Opal: Offline primitive discovery for accelerating offline reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2021.

Argenson, A. and Dulac-Arnold, G. Model-based offline planning. In *International Conference on Learning Representations (ICLR)*, 2021.

Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

Ball, P. J., Parker-Holder, J., Pacchiano, A., Choromanski, K., and Roberts, S. J. Ready policy one: World building through active learning. In *International Conference on Machine Learning (ICML)*, 2020.

Barber, D. and Agakov, F. The IM algorithm: a variational approach to information maximization. In *Neural Information Processing Systems (NeurIPS)*, 2003.

Berseth, G., Geng, D., Devin, C., Rhinehart, N., Finn, C., Jayaraman, D., and Levine, S. Smirl: Surprise minimizing reinforcement learning in unstable environments. In *International Conference on Learning Representations (ICLR)*, 2021.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym. *ArXiv*, abs/1606.01540, 2016.

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Burda, Y., Edwards, H., Storkey, A. J., and Klimov, O. Exploration by random network distillation. In *International Conference on Learning Representations (ICLR)*, 2019.

Campos Camúñez, V., Trott, A., Xiong, C., Socher, R., Giró Nieto, X., and Torres Viñals, J. Explore, discover and learn: unsupervised discovery of state-covering skills. In *International Conference on Machine Learning (ICML)*, 2020.

Castro, P. S. Scalable methods for computing state similarity in deterministic markov decision processes. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Clavera, I., Rothfuss, J., Schulman, J., Fujita, Y., Asfour, T., and Abbeel, P. Model-based reinforcement learning via meta-policy optimization. In *Conference on Robot Learning (CoRL)*, 2018.

Deisenroth, M. P. and Rasmussen, C. E. Pilco: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, 2011.

Draeger, A., Engell, S., and Ranke, H. D. Model predictive control using neural networks. *IEEE Control Systems Magazine*, 15:61–66, 1995.

Ebert, F., Finn, C., Dasari, S., Xie, A., Lee, A. X., and Levine, S. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *ArXiv*, abs/1812.00568, 2018.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations (ICLR)*, 2019.

Eysenbach, B., Salakhutdinov, R., and Levine, S. Robust predictable control. In *Neural Information Processing Systems (NeurIPS)*, 2021.

Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E., and Levine, S. Model-based value expansion for efficient model-free reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018.

Gelada, C., Kumar, S., Buckman, J., Nachum, O., and Bellemare, M. G. Deepmdp: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning (ICML)*, 2019.

Gregor, K., Rezende, D. J., and Wierstra, D. Variational intrinsic control. *ArXiv*, abs/1611.07507, 2016.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, 2018a.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. Soft actor-critic algorithms and applications. *ArXiv*, abs/1812.05905, 2018b.

Hafner, D., Lillicrap, T. P., Fischer, I. S., Villegas, R., Ha, D. R., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning (ICML)*, 2019.

Hafner, D., Lillicrap, T. P., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations (ICLR)*, 2020.

Hansen, N., Wang, X., and Su, H. Temporal difference learning for model predictive control. In *International Conference on Machine Learning (ICML)*, 2022.

Hasselt, H. V., Hessel, M., and Aslanides, J. When to use parametric models in reinforcement learning? In *Neural Information Processing Systems (NeurIPS)*, 2019.

Heess, N. M. O., Wayne, G., Silver, D., Lillicrap, T. P., Erez, T., and Tassa, Y. Learning continuous control policies by stochastic value gradients. In *Neural Information Processing Systems (NeurIPS)*, 2015.

Hernandaz, E. P. S. and Arkun, Y. Neural network modeling and an extended dmc algorithm to control nonlinear systems. In *American Control Conference*, 1990.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., Turck, F. D., and Abbeel, P. Vime: Variational information maximizing exploration. In *Neural Information Processing Systems (NeurIPS)*, 2016.

Jafferjee, T., Imani, E., Talvitie, E. J., White, M., and Bowling, M. Hallucinating value: A pitfall of dyna-style planning with imperfect environment models. *ArXiv*, abs/2006.04363, 2020.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Kang, K., Gradu, P., Choi, J. J., Janner, M., Tomlin, C. J., and Levine, S. Lyapunov density models: Constraining distribution shift in learning-based control. In *International Conference on Machine Learning (ICML)*, 2022.

Kidambi, R., Rajeswaran, A., Netrapalli, P., and Joachims, T. Morel : Model-based offline reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2020.

Kim, J., Park, S., and Kim, G. Unsupervised skill discovery with bottleneck option learning. In *International Conference on Machine Learning (ICML)*, 2021.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. Model-ensemble trust-region policy optimization. In *International Conference on Learning Representations (ICLR)*, 2018.

Laskin, M., Liu, H., Peng, X. B., Yarats, D., Rajeswaran, A., and Abbeel, P. Unsupervised reinforcement learning with contrastive intrinsic control. In *Neural Information Processing Systems (NeurIPS)*, 2022.

Lee, L., Eysenbach, B., Parisotto, E., Xing, E. P., Levine, S., and Salakhutdinov, R. Efficient exploration via state marginal matching. *ArXiv*, abs/1906.05274, 2019.

Lenz, I., Knepper, R. A., and Saxena, A. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems (RSS)*, 2015.

Li, L., Walsh, T. J., and Littman, M. L. Towards a unified theory of state abstraction for mdps. In *International Symposium on Artificial Intelligence and Mathematics*, 2006.

Liu, H. and Abbeel, P. Behavior from the void: Unsupervised active pre-training. In *Neural Information Processing Systems (NeurIPS)*, 2021.

Lu, K., Grover, A., Abbeel, P., and Mordatch, I. Reset-free lifelong learning with skill-space planning. In *International Conference on Learning Representations (ICLR)*, 2021.

Machado, M. C., Bellemare, M. G., and Bowling, M. A laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2017.

Mendonca, R., Rybkin, O., Daniilidis, K., Hafner, D., and Pathak, D. Discovering and achieving goals via world models. In *Neural Information Processing Systems (NeurIPS)*, 2021.

Nachum, O., Gu, S. S., Lee, H., and Levine, S. Data-efficient hierarchical reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Nachum, O., Gu, S. S., Lee, H., and Levine, S. Near-optimal representation learning for hierarchical reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2019.

Nagabandi, A., Konolige, K., Levine, S., and Kumar, V. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning (CoRL)*, 2019.

Nguyen, T. D., Shu, R., Pham, T., Bui, H. H., and Ermon, S. Temporal predictive coding for model-based planning in latent space. In *International Conference on Machine Learning (ICML)*, 2021.

Park, S., Choi, J., Kim, J., Lee, H., and Kim, G. Lipschitz-constrained unsupervised skill discovery. In *International Conference on Learning Representations (ICLR)*, 2022.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, 2017.

Pathak, D., Gandhi, D., and Gupta, A. K. Self-supervised exploration via disagreement. In *International Conference on Machine Learning (ICML)*, 2019.

Pong, V. H., Dalal, M., Lin, S., Nair, A., Bahl, S., and Levine, S. Skew-Fit: State-covering self-supervised reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2020.

Poole, B., Ozair, S., van den Oord, A., Alemi, A. A., and Tucker, G. On variational bounds of mutual information. In *International Conference on Machine Learning (ICML)*, 2019.

Rajeswar, S., Mazzaglia, P., Verbelen, T., Pich'e, A., Dhoedt, B., Courville, A. C., and Lacoste, A. Unsupervised model-based pre-training for data-efficient control from pixels. *ArXiv*, abs/2209.12016, 2022.

Rajeswaran, A., Ghotra, S., Levine, S., and Ravindran, B. Epopt: Learning robust neural network policies using model ensembles. In *International Conference on Learning Representations (ICLR)*, 2017.

Rasmussen, C. E. and Kuss, M. Gaussian processes in reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2003.

Rhinehart, N., Wang, J., Berseth, G., Co-Reyes, J. D., Hafner, D., Finn, C., and Levine, S. Information is power: Intrinsic control via information capture. In *Neural Information Processing Systems (NeurIPS)*, 2021.

Ross, S. and Bagnell, J. A. Agnostic system identification for model-based reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2012.

Salter, S., Wulfmeier, M., Tirumala, D., Heess, N. M. O., Riedmiller, M. A., Hadsell, R., and Rao, D. Mo2: Model-based offline options. In *Conference on Lifelong Learning Agents (CoLLAs)*, 2022.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. Mastering Atari, Go, Chess and Shogi by planning with a learned model. *Nature*, 588 7839:604–609, 2020.

Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., and Pathak, D. Planning to explore via self-supervised world models. In *International Conference on Machine Learning (ICML)*, 2020.

Sharma, A., Gu, S., Levine, S., Kumar, V., and Hausman, K. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations (ICLR)*, 2020.

Shyam, P., Jaśkowski, W., and Gomez, F. J. Model-based active exploration. In *International Conference on Machine Learning (ICML)*, 2019.

Sikchi, H. S., Zhou, W., and Held, D. Learning off-policy with online planning. In *Conference on Robot Learning (CoRL)*, 2022.

Stolle, M. and Precup, D. Learning options in reinforcement learning. In *Symposium on Abstraction, Reformulation and Approximation*, 2002.

Strouse, D., Baumli, K., Warde-Farley, D., Mnih, V., and Hansen, S. S. Learning more skills through optimistic exploration. In *International Conference on Learning Representations (ICLR)*, 2022.

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

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

Testud, J., Richalet, J., Rault, A., and Papon, J. Model predictive heuristic control: Applications to industial processes. *Automatica*, 14(5):413–428, 1978.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N. M. O., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2017.

Watter, M., Springenberg, J. T., Boedecker, J., and Riedmiller, M. A. Embed to control: A locally linear latent dynamics model for control from raw images. In *Neural Information Processing Systems (NeurIPS)*, 2015.

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

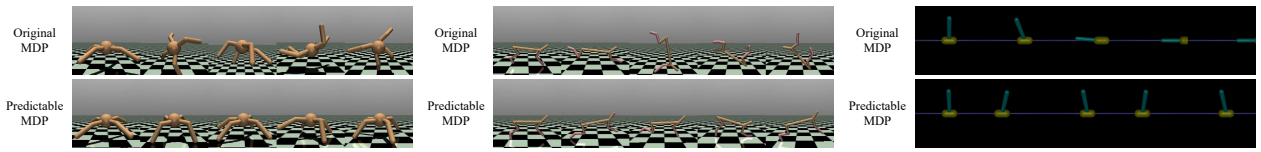
Wu, P., Escontrela, A., Hafner, D., Goldberg, K., and Abbeel, P. Daydreamer: World models for physical robot learning. In *Conference on Robot Learning (CoRL)*, 2022.

Wulfmeier, M., Rao, D., Hafner, R., Lampe, T., Abdolmaleki, A., Hertweck, T., Neunert, M., Tirumala, D., Siegel, N., Heess, N. M. O., and Riedmiller, M. A. Data-efficient hindsight off-policy option learning. In *International Conference on Machine Learning (ICML)*, 2021.

Xie, K., Bharadhwaj, H., Hafner, D., Garg, A., and Shkurti, F. Latent skill planning for exploration and transfer. In *International Conference on Learning Representations (ICLR)*, 2020.

Yarats, D., Fergus, R., Lazaric, A., and Pinto, L. Reinforcement learning with prototypical representations. In *International Conference on Machine Learning (ICML)*, 2021.

Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J. Y., Levine, S., Finn, C., and Ma, T. Mopo: Model-based offline policy optimization. In *Neural Information Processing Systems (NeurIPS)*, 2020.



(a) Ant (b) HalfCheetah (c) InvertedPendulum

Figure 10. Examples of PMA. PMA prevents unpredictable, chaotic actions so that every transition in the latent MDP is maximally predictable. Videos are available at https://seohong.me/projects/pma/.

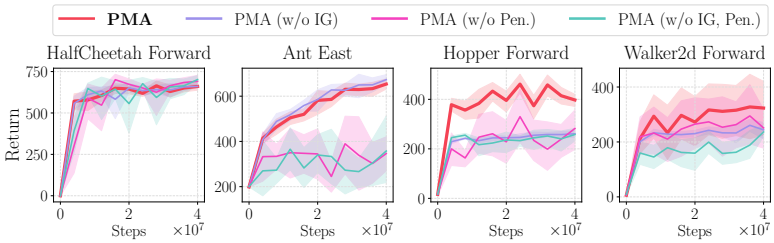


Figure 11. Ablation study of the disagreement bonus ("IG") during unsupervised training and disagreement penalty ("Pen.") during planning. The disagreement penalty generally stabilizes training, and the disagreement bonus improves performance.

A. Examples of PMA

To illustrate the difference between original MDPs and the corresponding predictable MDPs, we present qualitiative examples of PMA in Figure 10. In Ant and HalfCheetah, our predictable MDP only allows actions that are easy to model yet diverse enough to solve downstream tasks, preventing unpredictable behaviors such as chaotic flipping. In InvertedPendulum, we find that most of the learned latent actions move the agent in different directions while maintaining balance, even without early termination, in order to make the transitions maximally predictable. Videos are available at https://seohong.me/projects/pma/.

B. Ablation study

To evaluate the relative importance of each component of PMA, we ablate the information gain (disagreement bonus) term during unsupervised training and the disagreement penalty during periodic MPPI planning, and report performances in Figure 11. While there are small performance differences between the settings in HalfCheetah, these components improve and stabilize the performances in the other more complex environments by encouraging exploration and preventing distributional shifts.

C. Approximating Information Gain with Ensemble Disagreement

In this section, we provide a justification for our use of ensemble disagreement as a way to approximate the information gain term $I(\boldsymbol{S}'; \boldsymbol{\Theta} | \boldsymbol{\mathcal{D}}, \boldsymbol{Z})$ in Equation (6). First, let the random variable $\hat{\boldsymbol{S}}' \sim \hat{p}_z(\cdot | \boldsymbol{S}, \boldsymbol{Z}; \boldsymbol{\Theta})$ denote the *predicted* state under a model with parameters $\boldsymbol{\Theta}$. Since $\boldsymbol{S}' \to \boldsymbol{\Theta} \to \hat{\boldsymbol{S}}'$ forms a Markov chain conditioned on $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{Z}$ in our Bayesian setting, we get the following lower bound by the data processing inequality:

$$I(\boldsymbol{S}'; \boldsymbol{\Theta} | \boldsymbol{\mathcal{D}}, \boldsymbol{Z}) \geq I(\boldsymbol{S}'; \hat{\boldsymbol{S}}' | \boldsymbol{\mathcal{D}}, \boldsymbol{Z}) \tag{14}$$

$$= H(\hat{\boldsymbol{S}}' | \boldsymbol{\mathcal{D}}, \boldsymbol{Z}) - H(\hat{\boldsymbol{S}}' | \boldsymbol{\mathcal{D}}, \boldsymbol{Z}, \boldsymbol{S}'). \tag{15}$$

Now, we approximate the model posterior with an ensemble of E predictive models, $\{\hat{p}_z(\boldsymbol{s}' | \boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\theta}_i)\}_{i \in [E]}$ with $p(\boldsymbol{\theta} | \mathcal{D}) = \frac{1}{E} \sum_i \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_i)$, where each model is represented as a conditional Gaussian with the mean given by a neural network and a unit diagonal covariance, $\boldsymbol{s}' \sim \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\theta}_i), I)$. The terms in Equation (15) measure the uncertainty in the predicted next state before and after observing the outcome $\boldsymbol{S}'$, respectively. Yet, it is still intractable because there is no closed-form formulation for the differential entropy of a mixture of Gaussian distributions. Hence, we further simplify these terms as follows. First, we assume that the second term in Equation (15) has a negligible effect on the objective, which roughly corresponds to assuming a low training error (*i.e.*, if we know the value of $\boldsymbol{S}'$ and we update the models, they should agree

on $\hat{\boldsymbol{S}}'$, or at least have similar error). This assumption might not hold in heteroskedastic environments but is otherwise very convenient. Next, we empirically substitute the first term in Equation (15) with the variance of the ensemble means with a coefficient β, $\mathbb{E}[\beta \cdot \mathrm{Tr}[\mathbb{V}_i[\boldsymbol{\mu}(\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\theta}_i)]]]$, based on the fact that they both are correlated to the uncertainty in $\hat{\boldsymbol{S}}'$ (Sekar et al., 2020): if the predictions of the ensemble models are very different from one another (*i.e.*, the variance is large), the marginal entropy of $\hat{\boldsymbol{S}}'$ will also be large, and vice versa. As a result, we get our approximation in Equation (10). We also refer to prior works (Shyam et al., 2019; Sekar et al., 2020; Ball et al., 2020; Strouse et al., 2022) for similar connections between ensemble disagreement and information gain.

D. Theoretical Results

D.1. Technical Lemmas

For an MDP $\mathcal{M} := (\mathcal{S}, \mathcal{A}, \mu, p, r)$[3] and a policy π, we first define the discounted state and state-action distributions, and state the bellman flow constraint lemma.

Definition D.1. (Discounted state distribution) $d^\pi(\boldsymbol{s}) := (1-\gamma)\sum_{t=0}^{\infty} \gamma^t P(\boldsymbol{s}_t = \boldsymbol{s} | \mu, p, \pi)$.

Definition D.2. (Discounted state-action distribution) $d^\pi(\boldsymbol{s}, \boldsymbol{a}) := \pi(\boldsymbol{a}|\boldsymbol{s}) d^\pi(\boldsymbol{s})$.

Lemma D.3. *(Bellman flow constraint)*

$$d^\pi(\boldsymbol{s}) = (1-\gamma)\mu(\boldsymbol{s}) + \gamma \sum_{\boldsymbol{s}^- \in \mathcal{S}, \boldsymbol{a}^- \in \mathcal{A}} p(\boldsymbol{s}|\boldsymbol{s}^-, \boldsymbol{a}^-) d^\pi(\boldsymbol{s}^-, \boldsymbol{a}^-). \tag{16}$$

Proof.

$$\begin{aligned}
d^\pi(\boldsymbol{s}) &= (1-\gamma)(P(\boldsymbol{s}_0 = \boldsymbol{s}) + \gamma P(\boldsymbol{s}_1 = \boldsymbol{s}) + \gamma^2 P(\boldsymbol{s}_2 = \boldsymbol{s}) + \cdots) && (17)\\
&= (1-\gamma)\mu(\boldsymbol{s}) + \gamma(1-\gamma)(P(\boldsymbol{s}_1 = \boldsymbol{s}) + \gamma P(\boldsymbol{s}_2 = \boldsymbol{s}) + \cdots) && (18)\\
&= (1-\gamma)\mu(\boldsymbol{s}) + \gamma(1-\gamma) \sum_{\boldsymbol{s}^-, \boldsymbol{a}^-} p(\boldsymbol{s}|\boldsymbol{s}^-, \boldsymbol{a}^-)(P(\boldsymbol{s}_0 = \boldsymbol{s}^-, \boldsymbol{a}_0 = \boldsymbol{a}^-) + \gamma P(\boldsymbol{s}_1 = \boldsymbol{s}^-, \boldsymbol{a}_1 = \boldsymbol{a}^-) + \cdots) && (19)\\
&= (1-\gamma)\mu(\boldsymbol{s}) + \gamma \sum_{\boldsymbol{s}^-, \boldsymbol{a}^-} p(\boldsymbol{s}|\boldsymbol{s}^-, \boldsymbol{a}^-) d^\pi(\boldsymbol{s}^-, \boldsymbol{a}^-). && (20)
\end{aligned}$$

□

Now, we consider two MDPs with different transition dynamics, $\mathcal{M}_1 := (\mathcal{S}, \mathcal{A}, r, \mu, p_1)$ and $\mathcal{M}_2 := (\mathcal{S}, \mathcal{A}, r, \mu, p_2)$, and two policies, π_1 and π_2. We denote the expected return of π as $J_\mathcal{M}(\pi) := \frac{1}{1-\gamma}\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim d^\pi(\boldsymbol{s},\boldsymbol{a})}[r(\boldsymbol{s}, \boldsymbol{a})]$ and the maximum reward as $R := \max_{\boldsymbol{s}\in\mathcal{S}, \boldsymbol{a}\in\mathcal{A}} r(\boldsymbol{s}, \boldsymbol{a})$. We can bound their performance difference as follows.

Lemma D.4. *If the total variation distances of the dynamics and the policies are bounded as*

$$\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim d_1^{\pi_1}(\boldsymbol{s},\boldsymbol{a})}[D_{\mathrm{TV}}(p_1(\cdot|\boldsymbol{s}, \boldsymbol{a}) \| p_2(\cdot|\boldsymbol{s}, \boldsymbol{a}))] \le \epsilon_m, \tag{21}$$

$$\mathbb{E}_{\boldsymbol{s}\sim d_1^{\pi_1}(\boldsymbol{s})}[D_{\mathrm{TV}}(\pi_1(\cdot|\boldsymbol{s}) \| \pi_2(\cdot|\boldsymbol{s}))] \le \epsilon_\pi, \tag{22}$$

their performance difference satisfies the following inequality:

$$|J_{\mathcal{M}_1}(\pi_1) - J_{\mathcal{M}_2}(\pi_2)| \le \frac{R}{(1-\gamma)^2}(2\gamma\epsilon_m + 2\epsilon_\pi). \tag{23}$$

[3] In our theoretical analyses, we assume that the state and action spaces are finite for simplicity.

Proof. We first bound the difference in their discounted state-action distributions.

$$\sum_{\boldsymbol{s}\in\mathcal{S},\boldsymbol{a}\in\mathcal{A}} |d_1^{\pi_1}(\boldsymbol{s},\boldsymbol{a}) - d_2^{\pi_2}(\boldsymbol{s},\boldsymbol{a})| \tag{24}$$

$$= \sum_{\boldsymbol{s}\in\mathcal{S},\boldsymbol{a}\in\mathcal{A}} |\pi_1(\boldsymbol{a}|\boldsymbol{s})d_1^{\pi_1}(\boldsymbol{s}) - \pi_2(\boldsymbol{a}|\boldsymbol{s})d_2^{\pi_2}(\boldsymbol{s})| \tag{25}$$

$$\leq \sum_{\boldsymbol{s}\in\mathcal{S},\boldsymbol{a}\in\mathcal{A}} |\pi_1(\boldsymbol{a}|\boldsymbol{s})d_1^{\pi_1}(\boldsymbol{s}) - \pi_2(\boldsymbol{a}|\boldsymbol{s})d_1^{\pi_1}(\boldsymbol{s})| + \sum_{\boldsymbol{s}\in\mathcal{S},\boldsymbol{a}\in\mathcal{A}} |\pi_2(\boldsymbol{a}|\boldsymbol{s})d_1^{\pi_1}(\boldsymbol{s}) - \pi_2(\boldsymbol{a}|\boldsymbol{s})d_2^{\pi_2}(\boldsymbol{s})| \tag{26}$$

$$= \sum_{\boldsymbol{s}\in\mathcal{S}} d_1^{\pi_1}(\boldsymbol{s}) \sum_{\boldsymbol{a}\in\mathcal{A}} |\pi_1(\boldsymbol{a}|\boldsymbol{s}) - \pi_2(\boldsymbol{a}|\boldsymbol{s})| + \sum_{\boldsymbol{s}\in\mathcal{S}} |d_1^{\pi_1}(\boldsymbol{s}) - d_2^{\pi_2}(\boldsymbol{s})| \tag{27}$$

$$\leq \sum_{\boldsymbol{s}\in\mathcal{S}} |d_1^{\pi_1}(\boldsymbol{s}) - d_2^{\pi_2}(\boldsymbol{s})| + 2\epsilon_\pi \tag{28}$$

$$= \gamma \sum_{\boldsymbol{s}\in\mathcal{S}} \left| \sum_{\boldsymbol{s}^-\in\mathcal{S},\boldsymbol{a}^-\in\mathcal{A}} \big(p_1(\boldsymbol{s}|\boldsymbol{s}^-,\boldsymbol{a}^-)d_1^{\pi_1}(\boldsymbol{s}^-,\boldsymbol{a}^-) - p_2(\boldsymbol{s}|\boldsymbol{s}^-,\boldsymbol{a}^-)d_2^{\pi_2}(\boldsymbol{s}^-,\boldsymbol{a}^-)\big) \right| + 2\epsilon_\pi \tag{29}$$

$$\leq \gamma \sum_{\boldsymbol{s}^-\in\mathcal{S},\boldsymbol{a}^-\in\mathcal{A},\boldsymbol{s}\in\mathcal{S}} \left|p_1(\boldsymbol{s}|\boldsymbol{s}^-,\boldsymbol{a}^-)d_1^{\pi_1}(\boldsymbol{s}^-,\boldsymbol{a}^-) - p_2(\boldsymbol{s}|\boldsymbol{s}^-,\boldsymbol{a}^-)d_2^{\pi_2}(\boldsymbol{s}^-,\boldsymbol{a}^-)\right| + 2\epsilon_\pi \tag{30}$$

$$\begin{aligned}\leq \gamma &\sum_{\boldsymbol{s}^-\in\mathcal{S},\boldsymbol{a}^-\in\mathcal{A},\boldsymbol{s}\in\mathcal{S}} \left|p_1(\boldsymbol{s}|\boldsymbol{s}^-,\boldsymbol{a}^-)d_1^{\pi_1}(\boldsymbol{s}^-,\boldsymbol{a}^-) - p_2(\boldsymbol{s}|\boldsymbol{s}^-,\boldsymbol{a}^-)d_1^{\pi_1}(\boldsymbol{s}^-,\boldsymbol{a}^-)\right| \\ &+ \gamma \sum_{\boldsymbol{s}^-\in\mathcal{S},\boldsymbol{a}^-\in\mathcal{A},\boldsymbol{s}\in\mathcal{S}} \left|p_2(\boldsymbol{s}|\boldsymbol{s}^-,\boldsymbol{a}^-)d_1^{\pi_1}(\boldsymbol{s}^-,\boldsymbol{a}^-) - p_2(\boldsymbol{s}|\boldsymbol{s}^-,\boldsymbol{a}^-)d_2^{\pi_2}(\boldsymbol{s}^-,\boldsymbol{a}^-)\right| + 2\epsilon_\pi\end{aligned} \tag{31}$$

$$\begin{aligned}= \gamma &\sum_{\boldsymbol{s}^-\in\mathcal{S},\boldsymbol{a}^-\in\mathcal{A}} d_1^{\pi_1}(\boldsymbol{s}^-,\boldsymbol{a}^-) \sum_{\boldsymbol{s}\in\mathcal{S}} \left|p_1(\boldsymbol{s}|\boldsymbol{s}^-,\boldsymbol{a}^-) - p_2(\boldsymbol{s}|\boldsymbol{s}^-,\boldsymbol{a}^-)\right| \\ &+ \gamma \sum_{\boldsymbol{s}^-\in\mathcal{S},\boldsymbol{a}^-\in\mathcal{A}} \left|d_1^{\pi_1}(\boldsymbol{s}^-,\boldsymbol{a}^-) - d_2^{\pi_2}(\boldsymbol{s}^-,\boldsymbol{a}^-)\right| + 2\epsilon_\pi\end{aligned} \tag{32}$$

$$\leq 2\gamma\epsilon_m + 2\epsilon_\pi + \gamma \sum_{\boldsymbol{s}^-\in\mathcal{S},\boldsymbol{a}^-\in\mathcal{A}} \left|d_1^{\pi_1}(\boldsymbol{s}^-,\boldsymbol{a}^-) - d_2^{\pi_2}(\boldsymbol{s}^-,\boldsymbol{a}^-)\right| \tag{33}$$

$$= 2\gamma\epsilon_m + 2\epsilon_\pi + \gamma \sum_{\boldsymbol{s}\in\mathcal{S},\boldsymbol{a}\in\mathcal{A}} |d_1^{\pi_1}(\boldsymbol{s},\boldsymbol{a}) - d_2^{\pi_2}(\boldsymbol{s},\boldsymbol{a})|\,, \tag{34}$$

which implies

$$\sum_{\boldsymbol{s}\in\mathcal{S},\boldsymbol{a}\in\mathcal{A}} |d_1^{\pi_1}(\boldsymbol{s},\boldsymbol{a}) - d_2^{\pi_2}(\boldsymbol{s},\boldsymbol{a})| \leq \frac{1}{1-\gamma}(2\gamma\epsilon_m + 2\epsilon_\pi), \tag{35}$$

where we use Lemma D.3 in Equation (29). Hence, we obtain

$$|J_{\mathcal{M}_1}(\pi_1) - J_{\mathcal{M}_2}(\pi_2)| = \frac{1}{1-\gamma} \left| \sum_{\boldsymbol{s}\in\mathcal{S},\boldsymbol{a}\in\mathcal{A}} (d_1^{\pi_1}(\boldsymbol{s},\boldsymbol{a}) - d_2^{\pi_2}(\boldsymbol{s},\boldsymbol{a}))r(\boldsymbol{s},\boldsymbol{a}) \right| \tag{36}$$

$$\leq \frac{R}{1-\gamma} \sum_{\boldsymbol{s}\in\mathcal{S},\boldsymbol{a}\in\mathcal{A}} |d_1^{\pi_1}(\boldsymbol{s},\boldsymbol{a}) - d_2^{\pi_2}(\boldsymbol{s},\boldsymbol{a})| \tag{37}$$

$$\leq \frac{R}{(1-\gamma)^2}(2\gamma\epsilon_m + 2\epsilon_\pi). \tag{38}$$

□

Equation (23) is the same bound as Lemma B.3 in Janner et al. (2019), but we use a milder assumption in Equation (22), which only assumes that the expectation (not the maximum) of the total variation distance between the policies is bounded.

D.2. MBRL Performance Bound

We first present the performance bound of a policy π in the original MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mu, p, r)$ and its model-based MDP (Janner et al., 2019). We denote the model-based MDP with a learned predictive model $\hat{p}$ as $\hat{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, \mu, \hat{p}, r)$. We assume that the model $\hat{p}$ is trained on a dataset $\mathcal{D}$, which is collected by a data-collecting policy $\pi_{\mathcal{D}}$.

Theorem D.5. *For any policy π, if the total variation distances between (*i*) the true dynamics p and the learned model $\hat{p}$ and (*ii*) the policy π and the data-collection policy $\pi_{\mathcal{D}}$ are bounded as*

$$\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim d^{\pi_{\mathcal{D}}}(\boldsymbol{s},\boldsymbol{a})}[D_{\mathrm{TV}}(p(\cdot|\boldsymbol{s},\boldsymbol{a})\|\hat{p}(\cdot|\boldsymbol{s},\boldsymbol{a}))] \leq \epsilon_m, \tag{39}$$

$$\mathbb{E}_{\boldsymbol{s}\sim d^{\pi_{\mathcal{D}}}(\boldsymbol{s})}[D_{\mathrm{TV}}(\pi(\cdot|\boldsymbol{s})\|\pi_{\mathcal{D}}(\cdot|\boldsymbol{s}))] \leq \epsilon_\pi, \tag{40}$$

the performance difference of π between $\mathcal{M}$ and $\hat{\mathcal{M}}$ satisfies the following inequality:

$$|J_{\mathcal{M}}(\pi) - J_{\hat{\mathcal{M}}}(\pi)| \leq \frac{R}{(1-\gamma)^2}(4\epsilon_\pi + 2\gamma\epsilon_m). \tag{41}$$

Proof. From Lemma D.4, we get

$$|J_{\mathcal{M}}(\pi) - J_{\hat{\mathcal{M}}}(\pi)| \leq |J_{\mathcal{M}}(\pi) - J_{\mathcal{M}}(\pi_{\mathcal{D}})| + |J_{\mathcal{M}}(\pi_{\mathcal{D}}) + J_{\hat{\mathcal{M}}}(\pi)| \tag{42}$$

$$\leq \frac{R}{(1-\gamma)^2}(2\epsilon_\pi) + \frac{R}{(1-\gamma)^2}(2\epsilon_\pi + 2\gamma\epsilon_m) \tag{43}$$

$$= \frac{R}{(1-\gamma)^2}(4\epsilon_\pi + 2\gamma\epsilon_m). \tag{44}$$

□

D.3. PMA Performance Bound

We provide the performance bound of our predictable MDP abstraction. We denote our predictable latent MDP as $\mathcal{M}_P := (\mathcal{S}, \mathcal{Z}, \mu, p_z, r)$ with the latent action space $\mathcal{Z}$ and the reward function $r(\boldsymbol{s}, \boldsymbol{z}) = \sum_a \pi_z(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z}) r(\boldsymbol{s}, \boldsymbol{a})$, and its model-based MDP as $\hat{\mathcal{M}}_P := (\mathcal{S}, \mathcal{Z}, \mu, \hat{p}_z, r)$. We assume that the model $\hat{p}_z$ is trained on a dataset $\mathcal{D}_P$ collected by a data-collecting policy $\pi_{\mathcal{D}_P}$ in the latent MDP.

For the performance bound of a policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ between the original MDP and the predictable model-based MDP, we independently tackle the performance losses caused by (*i*) MDP abstraction and (*ii*) model learning. For the first part, we take a similar approach to Nachum et al. (2019); Ajay et al. (2021). For any $\boldsymbol{s} \in \mathcal{S}$, $\boldsymbol{a} \in \mathcal{A}$, and a probability distribution $\phi : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{Z})$, we define the following state distribution in $\mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$:

$$p_z^\phi(\cdot|\boldsymbol{s}, \boldsymbol{a}) := \sum_{\boldsymbol{z}} \phi(\boldsymbol{z}|\boldsymbol{s}, \boldsymbol{a}) p_z(\cdot|\boldsymbol{s}, \boldsymbol{z}). \tag{45}$$

Also, we define the optimal $\boldsymbol{z}$ distribution that best mimics the original transition distribution:

$$\phi^*(\cdot|\boldsymbol{s}, \boldsymbol{a}) := \operatorname*{arg\,min}_{\phi \in \mathcal{S}\times\mathcal{A}\to\mathcal{P}(\mathcal{Z})} D_{\mathrm{TV}}(p(\cdot|\boldsymbol{s}, \boldsymbol{a})\|p_z^\phi(\cdot|\boldsymbol{s}, \boldsymbol{a})). \tag{46}$$

For a policy π in the original MDP, we define its corresponding optimal latent policy as follows:

$$\pi_z^{\phi^*}(\cdot|\boldsymbol{s}) := \sum_{\boldsymbol{a}\in\mathcal{A}} \pi(\boldsymbol{a}|\boldsymbol{s}) \phi^*(\cdot|\boldsymbol{s}, \boldsymbol{a}). \tag{47}$$

Intuitively, this policy produces latent action distributions that mimic the next state distributions of π as closely as possible. Now, we state the performance bound of π between the original MDP and the predictable latent MDP.

Lemma D.6. *For any policy π, if the total variation distance between the original transition dynamics and the optimal latent dynamics is bounded as*

$$\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim d^{\pi}(\boldsymbol{s},\boldsymbol{a})}[D_{\mathrm{TV}}(p(\cdot|\boldsymbol{s}, \boldsymbol{a})\|p_z^{\phi^*}(\cdot|\boldsymbol{s}, \boldsymbol{a}))] \leq \epsilon_a, \tag{48}$$

the performance difference between the original MDP and the predictable latent MDP satisfies the following inequality:

$$|J_{\mathcal{M}}(\pi) - J_{\mathcal{M}_P}(\pi_z^{\phi^*})| \leq \frac{2R\gamma\epsilon_a}{(1-\gamma)^2}. \tag{49}$$

Proof. The next state distribution of $\pi_z^{\phi^*}$ at state $\boldsymbol{s} \in \mathcal{S}$ in the latent MDP can be written as follows:

$$p_z(\cdot|\boldsymbol{s}) = \sum_{\boldsymbol{z}\in\mathcal{Z}} \pi_z^{\phi^*}(\boldsymbol{z}|\boldsymbol{s}) p_z(\cdot|\boldsymbol{s},\boldsymbol{z}) \tag{50}$$

$$= \sum_{\boldsymbol{z}\in\mathcal{Z}} \sum_{\boldsymbol{a}\in\mathcal{A}} \pi(\boldsymbol{a}|\boldsymbol{s}) \phi^*(\boldsymbol{z}|\boldsymbol{s},\boldsymbol{a}) p_z(\cdot|\boldsymbol{s},\boldsymbol{z}) \tag{51}$$

$$= \sum_{\boldsymbol{a}\in\mathcal{A}} \pi(\boldsymbol{a}|\boldsymbol{s}) \sum_{\boldsymbol{z}\in\mathcal{Z}} \phi^*(\boldsymbol{z}|\boldsymbol{s},\boldsymbol{a}) p_z(\cdot|\boldsymbol{s},\boldsymbol{z}) \tag{52}$$

$$= \sum_{\boldsymbol{a}\in\mathcal{A}} \pi(\boldsymbol{a}|\boldsymbol{s}) p_z^{\phi^*}(\cdot|\boldsymbol{s},\boldsymbol{a}). \tag{53}$$

Hence, $J_{\mathcal{M}_P}(\pi_z^{\phi^*})$ is equal to $J_{\mathcal{M}^{\phi^*}}(\pi)$, where $\mathcal{M}^{\phi^*}$ is defined as $(\mathcal{S}, \mathcal{A}, \mu, p_z^{\phi^*}, r)$. Then, Equation (49) follows from Lemma D.4. □

Next, we bound the performance difference of $\pi_z^{\phi^*}$ between the latent MDP and latent model-based MDP.

Lemma D.7. *If the total variation distances between* (i) *the true dynamics p_z and the learned model $\hat{p}_z$ and* (ii) *the policy $\pi_z^{\phi^*}$ and the data-collection policy $\pi_{\mathcal{D}_P}$ are bounded as*

$$\mathbb{E}_{(\boldsymbol{s},\boldsymbol{z})\sim d^{\pi_{\mathcal{D}_P}}(\boldsymbol{s},\boldsymbol{z})}[D_{\mathrm{TV}}(p_z(\cdot|\boldsymbol{s},\boldsymbol{z}) \| \hat{p}_z(\cdot|\boldsymbol{s},\boldsymbol{z}))] \le \epsilon'_m, \tag{54}$$

$$\mathbb{E}_{\boldsymbol{s}\sim d^{\pi_{\mathcal{D}_P}}(\boldsymbol{s})}[D_{\mathrm{TV}}(\pi_z^{\phi^*}(\cdot|\boldsymbol{s}) \| \pi_{\mathcal{D}_P}(\cdot|\boldsymbol{s}))] \le \epsilon'_\pi, \tag{55}$$

the performance difference of $\pi_z^{\phi^*}$ between $\mathcal{M}_P$ and $\hat{\mathcal{M}}_P$ satisfies the following inequality:

$$|J_{\mathcal{M}_P}(\pi_z^{\phi^*}) - J_{\hat{\mathcal{M}}_P}(\pi_z^{\phi^*})| \le \frac{R}{(1-\gamma)^2}(4\epsilon'_\pi + 2\gamma\epsilon'_m). \tag{56}$$

Proof. The proof is the same as Theorem D.5. □

Now, we get the following performance bound of PMA:

Theorem D.8. *(PMA performance bound) If the abstraction loss, the model error, and the policy difference are bounded as follows:*

$$\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim d^{\pi}(\boldsymbol{s},\boldsymbol{a})}[D_{\mathrm{TV}}(p(\cdot|\boldsymbol{s},\boldsymbol{a}) \| p_z^{\phi^*}(\cdot|\boldsymbol{s},\boldsymbol{a}))] \le \epsilon_a, \tag{57}$$

$$\mathbb{E}_{(\boldsymbol{s},\boldsymbol{z})\sim d^{\pi_{\mathcal{D}_P}}(\boldsymbol{s},\boldsymbol{z})}[D_{\mathrm{TV}}(p_z(\cdot|\boldsymbol{s},\boldsymbol{z}) \| \hat{p}_z(\cdot|\boldsymbol{s},\boldsymbol{z}))] \le \epsilon'_m, \tag{58}$$

$$\mathbb{E}_{\boldsymbol{s}\sim d^{\pi_{\mathcal{D}_P}}(\boldsymbol{s})}[D_{\mathrm{TV}}(\pi_z^{\phi^*}(\cdot|\boldsymbol{s}) \| \pi_{\mathcal{D}_P}(\cdot|\boldsymbol{s}))] \le \epsilon'_\pi, \tag{59}$$

the performance difference of π between the original MDP and the predictable latent model-based MDP is bounded as:

$$|J_{\mathcal{M}}(\pi) - J_{\hat{\mathcal{M}}_P}(\pi_z^{\phi^*})| \le \frac{R}{(1-\gamma)^2}(2\gamma\epsilon_a + 4\epsilon'_\pi + 2\gamma\epsilon'_m). \tag{60}$$

Proof. From Lemma D.6 and Lemma D.7, we obtain

$$|J_{\mathcal{M}}(\pi) - J_{\hat{\mathcal{M}}_P}(\pi_z^{\phi^*})| \le |J_{\mathcal{M}}(\pi) - J_{\mathcal{M}_P}(\pi_z^{\phi^*})| + |J_{\mathcal{M}_P}(\pi_z^{\phi^*}) - J_{\hat{\mathcal{M}}_P}(\pi_z^{\phi^*})| \tag{61}$$

$$\le \frac{R}{(1-\gamma)^2}(2\gamma\epsilon_a) + \frac{R}{(1-\gamma)^2}(4\epsilon'_\pi + 2\gamma\epsilon'_m) \tag{62}$$

$$= \frac{R}{(1-\gamma)^2}(2\gamma\epsilon_a + 4\epsilon'_\pi + 2\gamma\epsilon'_m). \tag{63}$$

□

E. Theoretical Comparison between PMA and DADS

In this section, we theoretically compare the objectives of PMA and DADS in terms of mutual information maximization, entropy approximation, and state coverage.

Mutual information maximization. We first compare the mutual information (MI) term $I(\boldsymbol{S}'; \boldsymbol{Z}|\boldsymbol{S})$ in the objectives of PMA and DADS. Here, we ignore the information gain term of PMA, *i.e.*, $\beta = 0$, which we will discuss later. Also, in order to simplify the analysis of information-theoretic objectives, we assume that the latent action space $\mathcal{Z}$ is discrete, *i.e.* $\mathcal{Z} = [Z]$. With these assumptions, the difference between PMA and DADS mainly lies in the exploration policy, $\pi_e(\boldsymbol{z}|\boldsymbol{s})$. While DADS first samples a latent action $\boldsymbol{z}$ at the beginning of each rollout ($\pi_e(\cdot|\boldsymbol{s}_0) = \text{Unif}(\mathcal{Z})$) and persists it throughout the entire episode ($\pi_e(\boldsymbol{z}_t = \boldsymbol{z}_{t-1}|\boldsymbol{s}_t) = 1$), PMA always uses a uniform random policy ($\pi_e(\cdot|\boldsymbol{s}) = \text{Unif}(\mathcal{Z})$) because we have assumed $\beta = 0$.

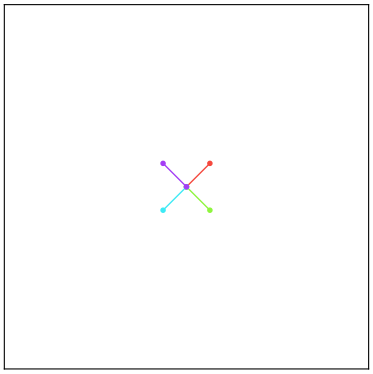


Figure 12. An example of the trajectories of a DADS policy ($Z = 4$).

We first consider the following upper bound of the MI objective, $I(\boldsymbol{S}'; \boldsymbol{Z}|\boldsymbol{S})$,

$$\max_{\pi_e, \pi_z} I(\boldsymbol{S}'; \boldsymbol{Z}|\boldsymbol{S}) = \max_{\pi_e, \pi_z} H(\boldsymbol{Z}|\boldsymbol{S}) - H(\boldsymbol{Z}|\boldsymbol{S}, \boldsymbol{S}') \tag{64}$$

$$\leq \max_{\pi_e, \pi_z} H(\boldsymbol{Z}|\boldsymbol{S}). \tag{65}$$

Equation (65) achieves its maximum of $\log Z$ when $p^{\pi_e, \pi_z}(\boldsymbol{z}|\boldsymbol{s})$ is a uniform random distribution, where $p^{\pi_e, \pi_z}(\boldsymbol{s}, \boldsymbol{z})$ is the state-latent action distribution from the policies. PMA uses $\pi_e(\cdot|\boldsymbol{s}) = \text{Unif}(\mathcal{Z})$, which precisely corresponds to this optimal condition. On the other hand, DADS's distribution can be rewritten as

$$p^{\pi_e, \pi_z}(\boldsymbol{z}|\boldsymbol{s}) = \frac{p^{\pi_e, \pi_z}(\boldsymbol{s}|\boldsymbol{z}) p^{\pi_e, \pi_z}(\boldsymbol{z})}{p^{\pi_e, \pi_z}(\boldsymbol{s})} \propto \frac{p^{\pi_e, \pi_z}(\boldsymbol{s}|\boldsymbol{z})}{p^{\pi_e, \pi_z}(\boldsymbol{s})}. \tag{66}$$

Unlike PMA, this does not necessarily correspond to a uniform distribution. For example, at the red dot state in Figure 12, we can see that $p^{\pi_e, \pi_z}(\boldsymbol{s}|\boldsymbol{z})$ is nonzero for the corresponding latent action but is zero for the other latent actions. This could make DADS suboptimal in terms of MI maximization.

Entropy approximation. A similar difference can be found in the approximation of the marginal entropy term $H(\boldsymbol{S}'|\boldsymbol{S})$. Both PMA and DADS use the following approximation:

$$H(\boldsymbol{S}'|\boldsymbol{S}) = \mathbb{E}\left[-\log p^{\pi_e, \pi_z}(\boldsymbol{s}'|\boldsymbol{s})\right] \tag{67}$$

$$= \mathbb{E}\left[-\log \int p^{\pi_e, \pi_z}(\boldsymbol{z}|\boldsymbol{s}) p^{\pi_e, \pi_z}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}) d\boldsymbol{z}\right] \tag{68}$$

$$\approx \mathbb{E}\left[-\log \int u(\boldsymbol{z}) p^{\pi_e, \pi_z}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}) d\boldsymbol{z}\right] \tag{69}$$

$$\approx \mathbb{E}\left[-\log \left(\frac{1}{L}\sum_{i=1}^{L} p^{\pi_e, \pi_z}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}_i)\right)\right], \tag{70}$$

where it approximates $p^{\pi_e, \pi_z}(\boldsymbol{z}|\boldsymbol{s})$ to a uniform distribution $u(\boldsymbol{z})$, and $\boldsymbol{z}_i$'s are sampled from $u(\cdot)$. While this approximation may not be accurate in DADS due to the same reason above, PMA satisfies $p^{\pi_e, \pi_z}(\boldsymbol{z}|\boldsymbol{s}) = u(\boldsymbol{z})$, making the approximation in the $\log$ exact.

State coverage. Another difference between PMA and DADS is the presence of the information gain term in the PMA objective. This is because the MI term alone does not necessarily cover the state space since MI is invariant to any invertible transformations of the input random variables, *i.e.*, $I(\boldsymbol{X}; \boldsymbol{Y}) = I(f(\boldsymbol{X}); g(\boldsymbol{Y}))$ for any random variables $\boldsymbol{X}$ and $\boldsymbol{Y}$, and invertible functions f and g. As a result, MI can be fully maximized with limited state coverage and does not necessarily encourage exploration (Campos Camúñez et al., 2020; Strouse et al., 2022; Park et al., 2022), which necessitates another term for maximizing the state-action coverage in PMA.

F. Training Procedures

F.1. PMA Training Procedure

PMA is trained with SAC (Haarnoja et al., 2018b). We describe several additional training details of PMA.

Algorithm 1 Predictable MDP Abstraction (PMA)

Initialize action decoder $\pi_z(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$, VLB predictive model $\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\phi})$, ensemble predictive models $\{\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\theta_i})\}$, (optional) exploration policy $\pi_e(\boldsymbol{z}|\boldsymbol{s})$, on-policy stochastic buffer $\mathcal{D}_S$, on-policy deterministic buffer $\mathcal{D}_D$, replay buffer $\mathcal{D}$
for $i \leftarrow 1$ to (# epochs) **do**
  for $j \leftarrow 1$ to (# steps per epoch)$/2$ **do**
    Sample latent action $\boldsymbol{z} \sim \pi_e(\boldsymbol{z}|\boldsymbol{s})$
    Sample action $\boldsymbol{a} \sim \pi_z(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$
    Add transition $(\boldsymbol{s}, \boldsymbol{z}, \boldsymbol{a}, \boldsymbol{s}')$ to $\mathcal{D}_S$, $\mathcal{D}$
  end for
  for $j \leftarrow 1$ to (# steps per epoch)$/2$ **do**
    Sample latent action $\boldsymbol{z} \sim \pi_e(\boldsymbol{z}|\boldsymbol{s})$
    Compute deterministic action $\boldsymbol{a} = \mathbb{E}[\pi_z(\cdot|\boldsymbol{s}, \boldsymbol{z})]$
    Add transition $(\boldsymbol{s}, \boldsymbol{z}, \boldsymbol{a}, \boldsymbol{s}')$ to $\mathcal{D}_D$
  end for
  Fit VLB predictive model using mini-batches from $\mathcal{D}_S$
  Fit ensemble predictive models using mini-batches from $\mathcal{D}_D$
  Train action decoder with $r(\boldsymbol{s}, \boldsymbol{z}, \boldsymbol{a}, \boldsymbol{s}') = \log \hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\phi}) - \log \frac{1}{L} \sum_{i=1}^{L} \hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}_i; \boldsymbol{\phi}) + \beta \cdot \mathrm{Tr}[\mathbb{V}_i[\boldsymbol{\mu}(\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\theta}_i)]]$ with SAC using mini-batches from $\mathcal{D}$
  (Optional) Train exploration policy π_e with SAC using mini-batches from $\mathcal{D}_S$
  Clear on-policy buffers $\mathcal{D}_S$, $\mathcal{D}_D$
end for

Replay buffer. Since we jointly train both the action decoder and the model, we need to be careful about using old, off-policy samples to train the components of PMA. While we can use old samples to train the action decoder $\pi_z(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$ as long as we recompute the intrinsic reward (because SAC is an off-policy algorithm), we cannot use old samples to train the predictive models or the exploration policy $\pi_e(\boldsymbol{z}|\boldsymbol{s})$. Hence, we use a replay buffer only for the action decoder, and train the other components with on-policy data.

Sampling strategy. PMA has two different kinds of predictive models: the VLB predictive model $\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\phi})$ to approximate $I(\boldsymbol{S}'; \boldsymbol{Z}|\boldsymbol{S})$, and an ensemble of E models $\{\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z}; \boldsymbol{\theta_i})\}$ to approximate $I(\boldsymbol{S}'; \boldsymbol{\Theta}|\boldsymbol{\mathcal{D}}, \boldsymbol{Z})$. While one may just simply use the mean of the ensemble outputs for the VLB predictive model, we find that it is helpful to train them separately with different sampling strategies. Specifically, we train the VLB predictive model with stochastic trajectories and the ensemble models with deterministic trajectories. Since we always use deterministic actions from the pre-trained action decoder during the test time, it is beneficial to have a latent predictive model trained from such deterministic actions. As such, at each epoch, we sample a half of the epoch transitions using deterministic actions to train the ensemble models and the other half using stochastic actions to train the other components. At test time, we use the mean of the ensemble model's outputs for model-based planning or model-based RL.

We summarize the full training procedure of PMA in Algorithm 1.

F.2. MPPI Training Procedure

MPPI (Williams et al., 2016) is a zeroth-order planning algorithm based on model predictive control (Testud et al., 1978), which finds an optimal action sequence via iterative refinement of randomly sampled actions. Specifically, at step t, MPPI aims to find the optimal (latent) action sequence $(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}, \dots, \boldsymbol{z}_{t+H-1})$ of length H via M iterations of refinement. At each iteration, MPPI samples N action sequences from a Gaussian distribution, $\boldsymbol{z}^{(i)}_{t:t+H-1} \sim \mathcal{N}(\boldsymbol{\mu}_{t:t+H-1}, \boldsymbol{\Sigma})$ for $i \in [N]$, where $\boldsymbol{\mu}_{t:t+H-1}$ is the current mean parameter and $\boldsymbol{\Sigma}$ is a fixed diagonal covariance matrix. It then computes the sum of the predicted rewards $\{R^{(i)} := \sum_{j=t}^{t+H-1} \hat{r}^{(i)}_j\}$ using the predicted states from the latent predictive model $\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z})$, and updates the mean parameter as follows:

$$\boldsymbol{\mu}_{t:t+H-1} \leftarrow \frac{\sum_{i \in [N]} e^{\alpha R^{(i)}} \boldsymbol{z}^{(i)}_{t:t+H-1}}{\sum_{i \in [N]} e^{\alpha R^{(i)}}}, \tag{71}$$

Algorithm 2 MPPI with PMA

Initialize mean parameter $\boldsymbol{\mu}_{0:T-1}$
for $t \leftarrow 0$ to $T-1$ **do**
  for $m \leftarrow 0$ to $M-1$ **do**
    Sample N latent action sequences $\boldsymbol{z}^{(i)}_{t:t+H-1} \sim \mathcal{N}(\boldsymbol{\mu}_{t:t+H-1}, \boldsymbol{\Sigma})$ for $i \in [N]$
    Compute sum of predicted rewards $\{R^{(i)} := \sum_{j=t}^{t+H-1} \hat{r}^{(i)}_j\}$ using predicted states from latent predictive model $\hat{p}_z(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{z})$
    Update $\boldsymbol{\mu}_{t:t+H-1}$ using Equation (71)
  end for
  Perform single action $\boldsymbol{a}_t = \mathbb{E}[\pi_z(\cdot|\boldsymbol{s}_t, \boldsymbol{z}_t)]$ with $\boldsymbol{z}_t = \boldsymbol{\mu}_t$ and get $\boldsymbol{s}_{t+1}$ from environment
end for

Algorithm 3 MBPO with PMA

Initialize task policy $\pi(\boldsymbol{z}|\boldsymbol{s})$, frozen replay buffer $\mathcal{D}_{\text{frozen}}$, replay buffer $\mathcal{D}$
for $i \leftarrow 1$ to (# epochs) **do**
  $d \leftarrow$ `TRUE`
  for $j \leftarrow 1$ to (# steps per epoch) **do**
    if $d =$ `TRUE` **then**
      Sample $\boldsymbol{s}$ from either $\mu(\cdot)$ with a probability of P or $\mathcal{D}_{\text{frozen}}$ with a probability of $(1-P)$
    end if
    Sample latent action $\boldsymbol{z} \sim \pi(\boldsymbol{z}|\boldsymbol{s})$
    Predict next state $\boldsymbol{s}' = \mathbb{E}[\hat{p}_z(\cdot|\boldsymbol{s}, \boldsymbol{z})]$
    Compute predicted reward r and predicted termination d using $\boldsymbol{s}$ and $\boldsymbol{s}'$
    if (current horizon length) $\geq H$ **then**
      $d \leftarrow$ `TRUE`
    end if
    Add transition $(\boldsymbol{s}, \boldsymbol{z}, r, \boldsymbol{s}')$ to $\mathcal{D}$
  end for
  Train task policy with SAC using mini-batches from $\mathcal{D}$
end for
for $i \leftarrow 1$ to (# evaluation rollouts) **do**
  while not termination **do**
    Compute latent action $\boldsymbol{z} = \mathbb{E}[\pi(\cdot|\boldsymbol{s})]$
    Compute action $\boldsymbol{a} = \mathbb{E}[\pi_z(\cdot|\boldsymbol{s}, \boldsymbol{s})]$
    Get r and $\boldsymbol{s}'$ from environment
  end while
end for

where α is a temperature hyperparameter. After M iterations of refinement, the agent performs only the first latent action and repeats this process to find the next optimal sequence $(\boldsymbol{z}_{t+1}, \boldsymbol{z}_{t+2}, \ldots, \boldsymbol{z}_{t+H})$. We summarize the full training procedure of MPPI in Algorithm 2.

F.3. MBPO Training Procedure

MBPO (Janner et al., 2019) is a Dyna-style (Sutton, 1991) model-based RL algorithm, which trains a model-free RL method on top of truncated model-based rollouts starting from intermediate environment states. In our zero-shot setting, MBPO uses the restored replay buffer from unsupervised training to sample starting states. Specifically, at each epoch, MBPO generates multiple model-based truncated trajectories $(\boldsymbol{s}_0, \boldsymbol{a}_0, r_0, \boldsymbol{s}_1, \boldsymbol{a}_1, r_1, \ldots, \boldsymbol{s}_{H-1}, \boldsymbol{a}_{H-1}, r_{H-1})$ of length H using the learned predictive model, where $\boldsymbol{s}_0$ is sampled either from the true initial state distribution $\mu(\cdot)$ with a probability of P or from the restored replay buffer $\mathcal{D}_{\text{frozen}}$ with a probability of $(1-P)$. It then updates the task policy $\pi(\boldsymbol{z}|\boldsymbol{s})$ using the collected trajectories with SAC (Haarnoja et al., 2018a), and repeats this process. Note that the restored replay buffer is only used to provide starting states. Also, if we set P to 1 and H to the original horizon length T, MBPO corresponds to vanilla SAC

trained purely on model-based transitions. After completing the training of MBPO, we measure the performance by testing the learned task policy in the true environment. We summarize the full training procedure of MBPO in Algorithm 3.

G. Implementation Details

We implement PMA on top of the publicly released codebase of LiSP (Lu et al., 2021). We release our implementation at the following repository: https://github.com/seohongpark/PMA. We run our experiments on an internal cluster consisting of A5000 or similar GPUs. Each run in our experiments takes no more than two days.

G.1. Environments

For our benchmark, we use seven MuJoCo robotics environments from OpenAI Gym (Todorov et al., 2012; Brockman et al., 2016): HalfCheetah, Ant, Hopper, Walker2d, InvertedPendulum, InvertedDoublePendulum, and Reacher. We mostly follow the environment configurations used in Sharma et al. (2020). We use an episode length of 200 for all environments. For HalfCheetah, Ant, Hopper, and Walker2d, we exclude the global coordinates from the input to the policy. For Hopper and Walker2d, we use an action repeat of 5 to have similar discretization time scales to the other environments. Regarding early termination, we follow the original environment configurations: Ant, Hopper, Walker2d, InvertedPendulum, and InvertedDoublePendulum have early termination conditions, while HalfCheetah and Reacher do not.

G.2. Tasks

We describe the reward functions of our 13 tasks. We mostly follow the reward scheme of the original task of each environment. For the environments with early termination, the agent additionally receives a reward of 1 at every step.

HalfCheetah Forward: The reward is the x-velocity of the agent.

HalfCheetah Backward: The reward is the negative of the x-velocity of the agent.

Ant East: The reward is the x-velocity of the agent.

Ant North: The reward is the y-velocity of the agent.

Hopper Forward: The reward is the x-velocity of the agent.

Hopper Hop: The reward is the maximum of 0 and the z-velocity of the agent.

Walker2d Forward: The reward is the x-velocity of the agent.

Walker2d Backward: The reward is the negative of the x-velocity of the agent.

InvertedPendulum Stay: The reward is the negative of the square of the x-position of the agent.

InvertedPendulum Forward: The reward is the x-velocity of the agent.

InvertedDoublePendulum Stay: The reward is the negative of the square of the x-position of the agent.

InvertedDoublePendulum Forward: The reward is the x-velocity of the agent.

Reacher Reach: The reward is the negative of the Euclidean distance between the target position and the fingertip.

G.3. Hyperparameters

We present the hyperparameters used in our experiments in Tables 1 to 3. For the MPPI results in Figure 5, we individually tune the MOPO penalty λ for each method and task (Table 2), where we consider $\lambda \in \{0, 1, 5, 20, 50\}$. For the MBPO results in Figure 6, we individually tune the MOPO penalty λ, the rollout horizon length H, and the reset probability P for each method and task (Table 3), where we consider $\lambda \in \{0, 1, 5, 20, 50\}$ and $(H, P) \in \{(1, 0), (5, 0), (15, 0), (15, 0.5), (50, 0), (50, 0.5), (200, 1)\}$. We use $(H, P) = (200, 1)$ for the SAC results in Figure 7. When training MBPO in Ant, we additionally apply $\max(0, \cdot)$ to the penalty-augmented reward to prevent the agent from preferring early termination to avoid negative rewards.

Table 1. Hyperparameters.

Hyperparameter	Value
# epochs	10000
# environment steps per epoch	4000
# gradient steps per epoch	64 (policy), 32 (model), 1 (RND network)
Episode length T	200
Minibatch size	256
Discount factor γ	0.995
Replay buffer size	100000
# hidden layers	2
# hidden units per layer	512
Nonlinearity	ReLU
Optimizer	Adam (Kingma & Ba, 2015)
Learning rate	3×10^{-4}
Target network smoothing coefficient τ	0.995
Ensemble size	5
Reward scale	10
Latent action dimensionality $\lvert\mathcal{Z}\rvert$	$\lvert\mathcal{A}\rvert$
PMA, DADS # samples for entropy approximation L	100
PMA ensemble variance scale β	5 (Walker2d), 50 (Hopper), 0.03 (Otherwise)
CM (Disag.) ensemble variance scale β	0.3 (Walker2d), 1 (HalfCheetah, Ant), 3 (Hopper), 10 (Otherwise)
MPPI horizon length H	15
MPPI population size N	256
MPPI # iterations M	10
MPPI temperature α	1
MPPI variance Σ	I

Table 2. MOPO λ for the MPPI results (Figure 5).

Task	PMA	DADS	CM (Rand.)	CM (Disag.)	CM (RND)
HalfCheetah Forward	1	1	1	1	1
HalfCheetah Backward	1	1	1	1	1
Ant East	20	20	20	20	20
Ant North	20	20	20	20	20
Hopper Forward	5	5	1	5	5
Hopper Hop	1	1	5	5	5
Walker2d Forward	1	1	1	1	1
Walker2d Backward	1	1	1	1	1
InvertedPendulum Stay	1	1	1	1	1
InvertedPendulum Forward	5	5	5	5	5
InvertedDoublePendulum Stay	0	5	5	5	5
InvertedDoublePendulum Forward	5	5	1	5	1
Reacher Reach	0	0	0	0	0

Table 3. (MOPO λ, MBPO H, MBPO P) for the MBPO results (Figure 6).

Task	PMA	DADS	CM (Rand.)	CM (Disag.)	CM (RND)
HalfCheetah Forward	$(1, 15, 0.5)$	$(1, 15, 0.5)$	$(1, 15, 0.5)$	$(1, 15, 0.5)$	$(1, 15, 0.5)$
Ant East	$(50, 15, 0.5)$	$(10, 15, 0.5)$	$(50, 15, 0.5)$	$(10, 15, 0.5)$	$(50, 15, 0.5)$
Hopper Forward	$(1, 200, 1)$	$(1, 200, 1)$	$(1, 200, 1)$	$(1, 50, 0.5)$	$(1, 15, 0.5)$
Walker2d Forward	$(1, 50, 0)$	$(1, 50, 0)$	$(1, 15, 0)$	$(1, 50, 0)$	$(1, 15, 0)$