Offline Model-based Adaptable Policy Learning for Decision-making in Out-of-Support Regions

Xiong-Hui Chen*, Fan-Ming Luo*, Yang Yu, Qingyang Li, Zhiwei Qin, Wenjie Shang, Jieping Ye

Abstract—In reinforcement learning, a promising direction to avoid online trial-and-error costs is learning from an offline dataset. Current offline reinforcement learning methods commonly learn in the policy space constrained to in-support regions by the offline dataset, in order to ensure the robustness of the outcome policies. Such constraints, however, also limit the potential of the outcome policies. In this paper, to release the potential of offline policy learning, we investigate the decision-making problems in out-of-support regions directly and propose offline Model-based Adaptable Policy LEarning (MAPLE). By this approach, instead of learning in in-support regions, we learn an adaptable policy that can adapt its behavior in out-of-support regions when deployed. We give a practical implementation of MAPLE via meta-learning techniques and ensemble model learning techniques. We conduct experiments on MuJoCo locomotion tasks with offline datasets. The results show that the proposed method can make robust decisions in out-of-support regions and achieve better performance than SOTA algorithms.

Index Terms—Offline reinforcement learning, model-based reinforcement learning, adaptable policy learning, meta learning.

1 Introduction

Recent studies have shown that reinforcement learning (RL) is a promising approach for real-world applications, e.g., sequential recommendation systems [1, 2, 3, 4] and robotic locomotion skill learning [5, 6]. However, the trial-and-error of RL in the real world [7] obstructs further applications in the scenarios where online data collection is expensive or dangerous, e.g., in robotics [8], healthcare [9], and recommender systems [10].

Offline (batch) RL learns a policy within a static dataset collected by a behavior policy without additional interactions with the environment [11, 12, 13, 14]. Since it avoids costly trial-and-error in real-world environments, offline RL is a promising way to handle the challenge in cost-sensitive applications. A significant challenge of offline RL is in answering counterfactual queries, which ask about how the performance (e.g., Q value) would have been if the agent were to execute an unseen action sequence, then learning to make optimal decisions based on the performance [11]. Fujimoto et al. [13] have shown that the distributional shift of states and actions, which comes from the discrepancy between evaluated policies and behavior policies, often leads to large extrapolation errors in value function estimation. In traditional model-free algorithms, the extrapolation error in value function estimation hurts the generalization performance of the learned policies. Since the additional samples, which can correct value estimation errors, are unavailable in the offline setting, the performance of learned policies based on value function is unstable [13].

On the other hand, model-based RL techniques, which learn dynamics models from collected datasets and learn the value function and policies based on the dynamics models, do not need to estimate the value functions relying on the collected datasets. However, similar challenges occur in dynamics model approximation. The dynamics model might overfit the limited dataset and suffer extrapolation errors in regions that behavior policies have not visited, which causes instability of the learned policy when deployment [15]. Here we call it out-of-support regions. Moreover, in model inference, the compounding error, that is, the accumulated prediction errors between simulation trajectories and reality, would be large even if the one-step prediction error is small [16, 17]. Recent studies in offline model-based RL [15, 18] have made significant progress in MuJoCo tasks [19]. These methods constrain policy sampling in dynamics models for robust policy learning. By using large penalty [15] or trajectory truncation [18, 15] in the regions with large prediction uncertainty (uncertainty is a designed metric to evaluate the confidence of prediction correctness) or compounding error, policy exploration is constrained in the regions of dynamics models where the predictions are corrected with high confidence, so as to avoid exploiting regions with risks of large extrapolation error. However, the constraints on dynamics models lead to a conservative policy learning process, which limits the potential of leveraging dynamics models: The visits on states and actions in out-of-support regions are more likely to be inhibited by the constraints, making the learned policy restrict the agent to be in similar regions as the behavior policy.

From the perspectives of counterfactual queries, we consider that model-based RL is promising to handle offline RL — ideal reconstructed dynamics models can simulate the transition dataset without the distributional-shift problem

* *Equal Contribution*

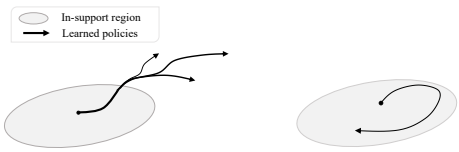
- *Xiong-Hui Chen, Fan-Ming Luo and Yang Yu are with the National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: {chenxh,luofm}@lamda.nju.edu.cn, yuy@nju.edu.cn.*
- *Qingyang Li is with DiDi Labs, California, United State. Zhiwei Qin is with Lyft Rideshare Labs. Work done while Zhiwei Qin, Wenjie Shang and JiepingYe were with DiDi Chuxing. E-mail: {qingyangli,qinzhiwei,shangwenjie,yejieping}@didiglobal.com.*
- *Corresponding author: Yang Yu*

given any policy, and the value function can be estimated via the "simulated" transition dataset directly. The bottleneck of offline model-based RL comes from the policy learning in the approximated dynamics model with extrapolation error. In this paper, instead of learning by tightly constraining policy exploration in in-support regions, we investigate decision-making in out-of-support regions directly. Finally, we propose a new offline policy learning framework, offline Model-based Adaptable Policy LEarning (MAPLE), to address the aforementioned issues. Ideally, MAPLE tries to model all possible transition dynamics in the out-of-support regions. Then an Adaptable policy is learned to be aware of each case to adapt its behavior to reach optimal performance. In the practical version of MAPLE, we use an ensemble technique to construct ensemble dynamics models. To be aware of each case of the transition dynamics and learn an adaptable policy, we use a meta-learning technique that introduces an extra environment-context extractor structure to represent dynamics patterns, and the policy adjusts itself according to the environment contexts.

We conduct experiments on the MuJoCo locomotion tasks. The results show that the sampling regions for robust offline policy learning can be extended by constructing transition patterns in out-of-support regions to cover the real case. The output adaptable policy yields better performance than SOTA algorithms when deployed. MAPLE gives a new direction to handle the offline policy learning problem in the dynamics models: Besides constraining on sampling and training dynamics models with better generalization, we can also model out-of-distribution regions by constructing all possible transition patterns.

2 Related Work

Reinforcement learning (RL) has shown to be a promising approach to complex real-world decision-making problems [1, 2, 3, 4]. However, unconstrained online trial-and-error in the training of RL agents prevents further applications of RL in safety-critical scenarios since it might result in large economic losses [11, 20, 21, 22]. Many studies propose to overcome the problem by offline (batch) RL algorithms [23]. Prior works on offline RL are based on model-free algorithms. To overcome the extrapolation error, which is introduced by the discrepancy between the offline dataset and true state-action distribution of learned target policies [13], these methods are designed to constrain target policies to be close to the behavior policies [13, 14, 24], to apply ensemble methods for robust value function estimation [25]. Nevertheless, such methods typically prevent the generalization of value functions beyond the offline data because of policy constraints. To handle the problem, recent studies also construct statistical confidence regions to obtain pessimism value functions based on the uncertainty estimation on out-of-support data, e.g., estimating the uncertainty based on the disagreement of bootstrapped Q-functions [26], or estimating the uncertainty based on the value function learned by perturbed rewards [27].



(a) learn to adapt (MAPLE) (b) learn by constraining

Fig. 1: Illustration of MAPLE compared with learning by constraining. The pointed lines represent the optimal trajectories of the learned policies. There are several policies in MAPLE since the method learns to adapt to multiple dynamics models. The gray oval represents the in-support region.

On the other hand, recent works also showed that policy learning with an approximated dynamics model has good potential to take robust actions outside the action distribution of behavior policies [18, 15]. The challenge comes from the extrapolation error of the dynamics models in out-of-support regions. To address the issues, these methods learn policies from dynamics models with uncertainty constraints. Uncertainty is a measure of prediction confidence on the next states. The uncertainty is often computed by the inconsistency in the ensemble dynamics model predictions for each state-action pair. Kidambi et al. [18] construct terminating states based on a hard threshold on uncertainty, while Yu et al. [15] use a soft reward penalty to incorporate uncertainty. The penalty constrains policy exploration and optimization to the regions with high consistency for better worst-case performance in the deployment environment [18, 15].

The difference between the aforementioned model-based methods and MAPLE is shown in Figure 1. Compared with previous methods [18, 15] learned by constraining (in Figure 1(b)), we learn to adapt to all possible dynamics transitions in the states in out-of-support regions (in Figure 1(a)). MAPLE uses a meta-learning technique of policy adaptation to design the practical algorithm of decision-making in out-of-support regions. Meta-learning [28] provides a way for policy adaptation. In meta-learning, a meta-policy model [29, 28] is learned from a set of source tasks and will adapt to a new environment with a small number of samples. Meta-based methods often need additional trajectories in the target environment to update its parameter. Online system identification (OSI) methods are to identify the environments automatically [30, 31, 32, 33, 34]. OSI methods always contain a unified policy (UP) and an environment-context extractor. The environment-context extractor extracts the information of the environment [31, 30, 33, 34]. The UP infers the action with the concatenation of the environment information and state as input. An environment-context extractor identifies the environment as the policy executing and does not need to update its parameter like meta-based methods. Thus, OSI techniques possess the ability to adapt to a new environment online.

There are also some studies on learning better dynamics models for offline model-based RL: Autoregressive dynamics model learning methods generate different dimensions of the next state and reward sequentially conditioned on previous dimensions for model learning in physics-based simulation environments [35]; Generative adversarial framework learns a dynamics model to generate a data distribution consistent with the real distribution to improve the generalization ability of model predictions to different policies [36, 37];

The generalization ability of the dynamics models can also be improved through a structured causal model [38].

3 Background and Notation

In the standard RL framework, an agent interacts with an environment governed by a Markov Decision Process (MDP) [39]. The agent learns a policy $\pi(a_t|s_t)$, which chooses an action $a_t \in \mathcal{A}$ given a particular state $s_t \in \mathcal{S}$, at each time-step $t \in \{0, 1, ..., T\}$, where T is the trajectory length. $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, respectively. The reward function $r_t = r(s_t, a_t) \in \mathbb{R}$ evaluates the immediate performance of the action a_t given the state s_t. The goal of RL is to find an optimal policy π^* that maximizes the multi-step cumulative discounted reward (i.e., long-term performance). The objective of RL is $\max_\pi J_\rho(\pi) := \mathbb{E}_{\tau\sim p(\tau|\pi,\rho)}\left[\sum_{k=0}^{T}\gamma^k r_k\right]$, where γ is the discount factor, and $p(\tau|\pi,\rho)$ is the probability of generating a trajectory $\tau := [s_0, a_0, ..., a_{T-1}, s_T]$ under the policy π and a dynamics model $\rho(s_{t+1}|s_t, a_t)$. In particular, $p(\tau \mid \pi) := d_0(s_0)\prod_{t=0}^{T-1}\rho(s_{t+1} \mid s_t, a_t)\pi(a_t|s_t)$, where $d_0(s_0)$ is the initial state distribution. A common way to find an optimal policy π^* is to optimize the policy with gradient ascent along $\nabla J_\rho(\pi)$ [39, 40].

In the offline RL setting, we are given only a static dataset $\mathcal{D} = \{(s_i, a_i, r_i, s_{i+1})\}$ collected by some unknown policy. The goal is to obtain a policy that maximizes J_ρ by only using the static dataset. In the standard scheme of offline model-based RL, we often learn a dynamics model $\hat{\rho}$ or a set of dynamics model $\hat{\rho} \in \mathcal{T}$ from $\mathcal{D}$ to predict the next states, then policies are learned based on the dynamics models. In this article, we follow the same setting as MOPO [15] where the reward functions are the same between policy learning and data collection in theoretical analysis. In practice, reward function models are learned to predict the rewards in the same manner as the dynamics model learning.

4 Offline Model-based Adaptable Policy Learning

We argue that in current offline model-based methods, the constraints of sampling to in-support regions of the dynamics model lead to a conservative policy learning process, limiting the potential of leveraging dynamics models. In this paper, to relax the constraints on the dynamics model, we investigate decision-making in out-of-support regions directly.

In this section, we first give a motivating example to show our ideal solution to out-of-support region decision-making (in Section 4.1). Then, we introduce a practical algorithm of the proposed solution for complex tasks, based on meta-learning techniques (in Section 4.2 and Section 4.3).

4.1 Decision-Making in Out-of-Support Regions

By rethinking the scheme of offline model-based RL, without loss of generality, we first formulate the problem as decision-making with a partially known dynamics model (Pak-DM) in a surrogate objective. In this problem, we have two dynamics models: a target dynamics model ρ and a partially known dynamics model ρ', where ρ is the deployment environment in the offline RL setting, and ρ' is used to approximate ρ. Due to the bias of data sampling in the offline setting and the limitation on the capacity of the function approximator, only in part of state-action space, we have $\rho'(s'|s,a) = \rho(s'|s,a)$, while the transitions in other parts of space are uncertain. We call the satisfied space "accessible space" (a.k.a., in-support regions) and its complement "inaccessible space" (a.k.a., out-of-support regions). In this problem, we assume the two subspaces have been predefined in some ways as in the offline setting (e.g., we can define the space in which an uncertainty quantification is larger than a threshold as the inaccessible space) and then discuss the decision-making problem in this setting. Formally, given an accessible space $\mathcal{X}_a$ and an inaccessible space $\mathcal{X}_i$, the partially known dynamics model is defined as:

$$\rho'(s'|s,a) = \begin{cases} \rho(s'|s,a) & [s,a] \in \mathcal{X}_a \\ \text{Unknown} & [s,a] \in \mathcal{X}_i \end{cases},$$

where $\mathcal{X}$ denotes the state-action concatenated space for brevity and $[s, a]$ denote a vector concatenating s and a. Our objective is to find a policy π^* to maximize J_ρ by only querying the partially known dynamics model ρ'. If $\mathcal{X}_i = \emptyset$, that is $\rho'(s'|s,a) = \rho(s'|s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$, the problem is reduced to a vanilla model-based policy learning problem with an oracle dynamics model. For simplification, we assume the oracle reward function r is given, but it can also be formulated as a partially known reward function in a similar way. Now we give an example of a Pak-DM in Figure 2.

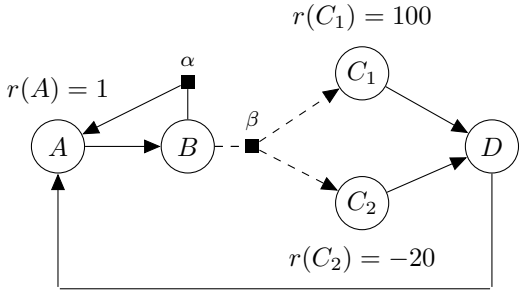


Fig. 2: An example of a Pak-DM. Each node denotes a state. Here we consider an MDP with finite state space, including A, B, C_1, C_2 and D. The directed edges denote the transition process. Here we consider a one-action transition in all states except for state B. On state B, we have action α and β, which are denoted as square nodes. By taking α on state B, the state will change to A. However, the transition after taking β in B is unknown. We use dashed directed edges to denote the possible transitions. In our formulation, edges to any node are valid. But we just consider the edges from B to C_1 and C_2 and omit the edges to A, B, and D for better readability. The reward function $r(s)$ will give a reward when the agent reaches A, C_1 or C_2.

In this setting, the model-based policy learning algorithms with constraints ([15, 18]) can be summarized as: finding the optimal policy without reaching inaccessible space. It might output a conservative policy since it avoids making decisions that might lead agents to out-of-support regions. Taking Figure 2 as an example, the output policy would be run in the loop of $A \to B \to \alpha \to A$. It will avoid

taking β on state B. If the real transition is $\rho(B,\beta) = C_2$, the policy can avoid the penalty -20 since C_2 will not be reached. On the other hand, while $\rho(B,\beta) = C_1$, the policy would miss the large bonus 100 in C_1.

To handle the above problem, in this article, we present a new offline policy learning method that utilizes a probe-reduce decision-making pipeline to improve policy generalization in out-of-support regions using the approximated ρ', which includes the following phases:

1) (Training) Construct a dynamics model set $\{\hat{\rho}_i\}$ via modeling all possible transitions in $\mathcal{X}_i$. (It is impractical to do that with infinite state space. We will give a practical solution in Section 4.3);
2) (Training) Learn the optimal policy from each model $\hat{\rho}_i$ to form the optimal policy set $\{\pi^*_{\hat{\rho}_i}\}$;
3) (Deployment) Initialize a state s_0 from the deployment environment ρ;
4) (Deployment) *Probe* the environment ρ by selecting an action a such that $[s,a] \in \mathcal{X}_i$. After getting the next state $s' = \rho(s'|s,a)$, store the tuple (s,a,s') in a memory (e.g., a replay buffer) $\mathcal{D}$. If there is no action a that allows $[s,a] \in \mathcal{X}_i$, randomly select a policy from the policy set to take an action.
5) (Deployment) *Reduce* the policy set by only keeping the policies whose corresponding transition model $\hat{\rho}_i$ can explain the experiences in the memory: $\{\hat{\rho}_i\} \leftarrow \{\rho \mid \rho(s'|s,a) = s', \forall (s,a,s') \in \mathcal{D}, \forall \rho \in \{\hat{\rho}_i\}\}$ and $\{\pi^*_{\hat{\rho}_i}\} \leftarrow \{\pi^*_\rho | \rho \in \{\hat{\rho}_i\}\}$;
6) (Deployment) Repeat Step 4 and 5 until the policy set is reduced to a single policy.

In this pipeline, we solve the decision-making problem in out-of-support regions by probing the uncertainty part of the deployment environment and adapting the policy for the environment. In Figure 2, the ideal MAPLE solution would construct two dynamics models: $\hat{\rho}_1$ where $\hat{\rho}_1(B,\beta) = C_1$, and $\hat{\rho}_2$ where $\hat{\rho}_2(B,\beta) = C_2$. Then we learn two optimal policies $\{\pi^*_{\rho_1}, \pi^*_{\rho_2}\}$ for each dynamics model. At deployment, we first randomly select a policy from the policy set to make decisions. Upon reaching B for the first time, where the transition on action β is uncertain, we take action β and get the next state. If the next state is C_1, the policy will reduce to $\pi^*_{\hat{\rho}_1}$, otherwise to $\pi^*_{\hat{\rho}_2}$. Therefore, if $\rho(B,\beta) = C_2$, the policy would initially run $A \to B \to \beta \to C_2 \to D \to A$ and then run in the loop of $A \to B \to \alpha \to A$ because the latter yields higher rewards. If $\rho(B,\beta) = C_1$, the policy would always run in the loop of $A \to B \to \beta \to C_1 \to D \to A$.

We then dive into the performance difference between the two pipelines for decision-making. In particular, we define a policy π_a which is learned by adapting through the above probing and reducing iterations, and a policy π_c which is learned by constraints: π_a firstly probe to visit state-action pair in inaccessible space based on the policies in the policy set until the policy set is reduced to a single policy. π_c is a conservative policy that avoids reaching any unknown regions. We give Theorem 1 to describe the performance gap $J_\rho(\pi_a) - J_\rho(\pi_c)$ between π_a and π_c. The full proof can be found in Appendix A.

Theorem 1. Given a target dynamics model ρ, a policy π_a learned by adapting, a policy π_c learned by constraints, and the maximum step N_m taken by π_a to probe and reduce the policy set to a single policy, the performance gap $J_\rho(\pi_a) - J_\rho(\pi_c)$ between π_a and π_c satisfies:

$$J_\rho(\pi_a) - J_\rho(\pi_c) \geq \Delta_c - \Delta_p - \gamma^{N_m+1} J_{\rho_\Delta}(\pi^*),$$

where Δ_c denotes the performance gap of the optimal policy π^* and π_c, while Δ_p denotes the performance degradation of MAPLE compared with π^* because of the phase of probing. $J_{\rho_\Delta}(\pi^*)$ denotes the performance degradation of π^* on the dynamics model ρ caused by different initial state distribution: $J_{\rho_\Delta}(\pi^*) = \mathbb{E}_{d^{\pi^\star}_{N_m+1}(s)}[V^\star(s)] - \mathbb{E}_{d^{\pi_a}_{N_m+1}(s)}[V^\star(s)]$, where $d^{\pi^\star}_{N_m+1}(s)$ and $d^{\pi_a}_{N_m+1}(s)$ denote the state distribution induced by $\pi^\star$ and π_a at the N_m+1 step and $V^*(s)$ denotes the expected long-term rewards of π^* at state s.

We can see that the performance gain of π_a is that it can automatically converge to the optimal policy after the loop of probing and reducing (i.e., Δ_c), while the cost of π_a comes from additional probing on inaccessible space, including less reward getting when probing (i.e., Δ_p) and a worse initial state distribution after probing (i.e., $J_{\rho_\Delta}(\pi^*)$).

Based on Theorem 1, we give the principles for choosing between the pipelines: Firstly, with a larger performance gap of Δ_c, π_a can reach a better performance than π_c. On the other hand, the tasks with large penalties on undesired behavior might make Δ_p larger, which reduces the overall performance of π_a. Besides, the tasks where sub-optimal behaviors easily lead agents to states with low value, e.g., unsafe states which are prone to terminate the trajectory, might make $J_{\rho_\Delta}(\pi^*)$ large, which also reduces the overall performance of π_a.

4.2 Efficient Decision-Making in Out-of-Support Regions with Meta-learning Techniques

It is computationally inefficient to learn optimal policies independently for each dynamics model since the policies' behaviors would be similar in in-support regions. For better efficiency, we introduce a context-aware adaptable policy, inspired by meta-learning techniques, to represent the set of learned policies. Here, we first introduce a new notation: the environment-context vector $z \in \mathcal{Z}$, where $\mathcal{Z}$ denotes the space of the context vectors. Given a set of dynamics models $\mathcal{T} := \{\hat{\rho}_i\}$, each $\hat{\rho}_i$ can be represented by a vector z. Formally, there is a mapping $\phi : \mathcal{T} \to \mathcal{Z}$. We call ϕ an environment-context extractor. The context-aware policy $\pi(a|s,z)$ takes actions based on the current state s and the vector of environment-context z of a given environment $z = \phi(\hat{\rho})$, where $\hat{\rho} \in \mathcal{T}$. We define the optimal environment-context extractor ϕ^* satisfying

$$\exists \pi_{\phi^*} \in \Pi, \forall \hat{\rho} \in \mathcal{T}, J_{\hat{\rho}}(\pi_{\phi^*}) = \max_\pi J_{\hat{\rho}}(\pi),$$

where $\pi_\phi := \pi(a_t|\phi(z_t|\rho_t), s_t)$ is an adaptable policy and Π denotes the policy class. It means that with the optimal extractor ϕ^*, we have a unified adaptable policy π_{ϕ^*}, which can make the optimal decisions in all of the environments in $\mathcal{T}$ based on the z inferred by ϕ^*. We will discuss the input of ϕ later.

In addition, we define the optimal adaptable policy $\pi^*_{\phi^*}$ to be one that satisfies $\forall \hat{\rho} \in \mathcal{T}, J_{\hat{\rho}}(\pi^*_{\phi^*}) = \max_\pi J_{\hat{\rho}}(\pi)$. With the optimal ϕ^* and $\pi^*_{\phi^*}$, and given any $\hat{\rho}$ in $\mathcal{T}$, the

adaptable policy can make the best decisions with the output of environment-context z. To achieve this, given a dynamics model set $\mathcal{T}$, we optimize ϕ and π_ϕ by the following objective function:

$$\phi^*, \pi^*_{\phi^*} = \arg\max_{\phi,\pi_\phi} \mathbb{E}_{\rho\sim\mathcal{T}}\left[J_\rho(\pi_\phi)\right], \tag{1}$$

where $\sim$ denotes a sample strategy to draw dynamics models $\hat{\rho}$ from the dynamics model set $\mathcal{T}$ s.t. $P[\hat{\rho}] > 0, \forall\hat{\rho} \in \mathcal{T}$. We take a uniform sampling strategy in the following analysis.

To learn the context z by $\phi(z|\hat{\rho})$, the main question is: What are suitable inputs to ϕ for context learning? In the robotics domain, similar environment contexts have been proposed recently [30, 41, 32]. The policy incorporates an online system identification module $\phi(z_t|s_t, \tau_{0:t})$, which utilizes the history of past states and actions $\tau_{0:t} = [s_0, a_0, ..., s_{t-1}, a_{t-1}, s_t]$ to predict the parameters of the dynamics in simulators. For example, τ could be a trajectory of robot interactions with varying friction coefficients, and z is the value of the coefficient. In practice, a recurrent neural network (RNN) is used to embed the sequential information into environment-context vectors $z_t = \phi(s_t, a_{t-1}, z_{t-1})$. In MAPLE, we follow the same structure to model the extractor but the trajectories are rollout in the constructed dynamics models. If the reward function is also partially known, r_{t-1} should be considered, that is $z_t = \phi(s_t, a_{t-1}, r_{t-1}, z_{t-1})$.

With an RNN-based environment-context extractor ϕ optimized with Equation (1), the context-aware policy π can automatically probe environments and reduce the policy set. Considering a given i-step partial trajectory $\tau_{0:i}$ and a subset of deterministic dynamics models $\mathcal{T}' \subset \mathcal{T}$ where the dynamics model $\hat{\rho} \in \mathcal{T}'$ is consistent with $\tau_{0:i}$, the objective from $i+1$ to T on given $\tau_{0:i}$ can be rewritten as $\mathbb{E}_{\hat{\rho}\sim\mathcal{T}'}[\mathbb{E}_{\tau\sim p(\pi,\hat{\rho})}[\sum_{k=i+1}^{T}\gamma^k r_k]]$. Since the dynamics models in $\mathcal{T}'$ are indistinguishable at step $i+1$, the optimal policy at this step would converge to a stochastic policy if the optimal actions are different among the dynamics models. If $\hat{\rho}$ is sampled uniformly from $\mathcal{T}'$ and the optimal cumulative rewards $\sum_{k=i+1}^{T}\gamma^k r^*_k$ are the same for each dynamics model, the optimal policy at $i+1$ is to uniformly-sampled actions from the optimal actions of each dynamics model. If the optimal cumulative rewards are different, the action probabilities are weighted by the cumulative rewards of each dynamics model. On the other hand, partial trajectories from different dynamics models might predict the same z. If the optimal actions in the same state are conflicting, to increase the performance of objective Equation (1), the policy gradient has to backpropagate from π to ϕ. If the partial trajectories $\tau_{0:i}$ are different, the contexts in these partial trajectories would be distinctive. Finally, the output action distribution of the context-aware policy would be optimized in a subset of the dynamics models in which the partial trajectories $\tau_{0:i}$ are consistent.

4.3 Practical Implementation of Offline Model-based Adaptable Policy Learning

From the decision-making problem with Pak-DM to the real offline setting, the additional thing we should consider is the recognition of the inaccessible space and the construction of the dynamics model set. Especially in tasks with infinite state-action space, it is impractical to find the exact inaccessible space and recover all possible transitions in it. As a practical implementation, we use the ensemble technique to learn the dynamics model set, which would predict similar transitions in the accessible space and tend to predict different transitions in inaccessible space. With large ensemble models with different initialized weights, we can construct a large number of transition cases in the inaccessible space. If the real transition pattern falls into the variant of the ensemble dynamics model set, relying on the interpolation ability of the environment-context extractor ϕ, the adaptable policy π can take appropriate actions. However, only relying on the randomness of the initialization, to cover real cases in all out-of-support region need sufficiently enough dynamics model set, which would be highly expensive. In order to trade off the cost of model construction and better adaptivity, in the practical implementation, we use several tricks to constrain the policy exploration in the ensemble dynamics model set: 1) We add some mild constraints to the exploration region. To mitigate the compounding error of the model, we constrain the maximum rollout length to a fixed number. Besides, at each step, a penalty is calculated according to the model uncertainty $U(s_t, a_t)$, which measures the prediction uncertainty of the learned transition models at (s_t, a_t). As a result, a reward provided to agents consists of two parts: a reward given by a reward function and a penalty calculated by $U(s_t, a_t)$. The constraints are the same as MOPO [15], but the coefficients are more relaxed; 2) As we increase the rollout length, the large compounding error might lead the agent to reach entirely unreal regions in which the states would never appear in the deployment environment. These samples

Algorithm 1 Offline model-based adaptable policy learning

Input: ϕ_φ as an environment-context extractor, parameterized by φ; Adaptable policy network π_θ parameterized by θ; Offline dataset $\mathcal{D}_{\text{off}}$; Trajectory termination rule f; Rollout horizon H;
Process:

Generate an ensemble of m dynamics models $\{\hat{\rho}_i\}$ via supervised learning
Initialize an empty buffer $\mathcal{D}_{\text{rollout}}$; Add hidden states z to each tuple in $\mathcal{D}_{\text{rollout}}$ and initialize with **0**
for 1, 2, 3, ... **do**
  Randomly select the dynamics model $\hat{\rho}_i$ in $\{\hat{\rho}_i\}$ and sample s_1, a_0, z_0 from $\mathcal{D}_{\text{off}}$
  for t=1, 2 , ..., H **do**
    Sample z_t from $\phi_\varphi(z|s_t, a_{t-1}, z_{t-1})$ and then sample a_t from $\pi_\theta(a|s_t, z_t)$
    Rollout one step $s_{t+1} \sim \hat{\rho}_i(s|s_t, a_t)$ and $r_{t+1} = r(s_t, a_t)$
    Compute the terminal state $d_{t+1} = f(s_{t+1})$
    Compute the reward penalty: $r_{t+1} \leftarrow r_{t+1} - \lambda U(s_t, a_t)$
    Add $(s_{t+1}, r_{t+1}, d_{t+1}, s_t, a_t, z_t)$ to $\mathcal{D}_{\text{rollout}}$
    Break the rollout if d_{t+1} is True
  end for
  Update the stored hidden states z in $\mathcal{D}_{\text{off}}$ with ϕ_φ
  Use SAC [42] to update φ and θ via Equation (1) with $\mathcal{D}_{\text{rollout}}$ and $\mathcal{D}_{\text{off}}$
end for

are useless for adaptable policy learning and might make the training process unstable. Given a task, we can construct some simple rules to discriminate the entirely unreal regions and terminate the trajectory rollout (i.e., set the "done" flag to True) when reaching these regions. In our implementation, we terminate trajectories when the predicted next states are out of range of $(-s_{\min}, s_{\max})$, where $s_{\min}$ and $s_{\max}$ are two hyper-parameters to define a reasonable range of state space.

Based on the above techniques, we propose the practical implementation of offline model-based adaptable policy learning in Algorithm 1. More details can be found in the Appendix B.

5 A Toy-Environment Verification

In this work, we give a probe-reduce pipeline for decision-making in out-of-support regions. In Section 4.2, we use an environment-context extractor and an adaptable policy to build the practical algorithm. We claim that with the RNN-based environment-context extractor ϕ optimized with Equation (1), the context-aware policy π can automatically probe environments and reduce the policy set. In this section, we design a toy environment, Treasure Explorer, to verify the argument. Figure 3 illustrates the Treasure-Explorer environment. Treasure Explorer follows the problem of decision-making with a partially known dynamics model (Pak-DM) in a surrogate objective. We learn an adaptable policy and a conservative policy in the environment and show the decision process of the adaptable policy matching the claims in the previous sections.

5.1 Verification Setup

In this environment, we only have a partially known dynamics model as illustrated in Figure 3 and would like to maximize the reward in a deployed environment. The deployed environment could be one of the three dynamics models: ghost, nothing or treasure, as illustrated in Figure 4(c). In the task, the agent is initialized at the lower-left room. By taking one of the available actions, the agent will move to the room pointed by the arrow. After moving into different rooms, the agent will hit the object in the room and get a reward, then get back to the lower-left room immediately. We set $r = 1$ when hitting the apple, $r = 10$ by hitting the treasure, and $r = -1$ by hitting the ghost. The agent can make decisions 5 times. That is, the horizon of a trajectory is set to 5.

We give the optimal policy and the ideal probe-reduce policy:

- **Optimal policy**: The optimal policy depends on the object in the inaccessible space. If the object in the inaccessible space gives -10 (ghost) or 0 (nothing) reward, the optimal policy is going right every time. If the object provided 10 reward (treasure), the optimal policy is going up forever.
- **Probe-reduce policy**: Ideally, a probe-reduce policy will first probe the inaccessible space: it will choose to go up and observe the object in the upper-left room. Based on the object obtained in the upper-left room, the possible dynamics model set will be reduced to one of the three models. Subsequently, it will choose the optimal policy to control the robot according to the confirmed dynamics model.

In Section 4.2, we propose to learn an adaptable policy based on meta-learning techniques to archive the efficient probe-reduce decision-making. In this section, we give the simplest implementation: We randomly select dynamics models from the dynamic model set (shown in Figure 4) with the same probability for each time, construct the environment-context extractor with a GRU-based recurrent neural network [43] and the adaptable policy with a multi-layer perceptron (MLP) and adopt PPO [44] to optimize the policy and the extractor. Besides, we also give the simplest implementation of conservative policy learning: we learn an MLP policy in the environment but with large penalties when the agent reaches an inaccessible space. The policy is also optimized with PPO.

5.2 Results

Figure 5 shows the learning curve of each method. There are two dash lines indicating the theoretical performance of the optimal policy and the probe-reduce policy. We can see that the performance of the adaptable policy can converge to the theoretical performance of the probe-reduce policy. In terms of the performance in the dynamics model set, the results demonstrate that the learned adaptable policy is consistent with the probe-reduce policy. The performance gap of the conservative policy and the adaptation ability in the training environments depend on the sampling strategy of the adaptable policy learning. With a larger probability of training with the treasure dynamics model, the return of adaptable policy would be larger.

In order to investigate the behavior patterns of the policies in different deployed environments, we record the up-to-now cumulative rewards at each step in a single trajectory. In particular, for the i-step, the up-to-now cumulative rewards is $\sum_{k=0}^{i} r_k$. The result is presented in Figure 6.

As can be seen in Figure 6, since the adaptable policy tries to probe the environment at the first step, the first-step reward is varied to the environments. After that, the RNN recognizes the environment, and the agent adjusts its policy to the optimal policy for the environment. As can be seen in Figure 6, after the first step, the slope of the up-to-now cumulative rewards of the adaptable policy are the same to the optimal policy. In contrast, the conservative agent always chooses to exploit the room with 1-reward. Besides, the cumulative rewards curve of the probe-reduce policy is the same as the adaptable policy.

We also present the trajectory sampled by the optimal policy, conservative policy, and adaptable policy in Appendix D. We can find the trajectories match the behaviors of the probe-reduce policy: probing first and then exploiting accordingly.

This result supports the claim in Section 4.2, the adaptable policies indeed possess such a probing-reduce characteristic. Without such adaptability, the agent is prone to stick at a local minimum solution and produces a conservative policy.

The results are also consistent with the Theorem 1. As can be found in the case of "Ghost", tasks with large penalties on undesirable behavior might make Δ_p larger, which reduces the overall performance of the adaptable policy. In this case, the performance of the adaptable policy is significantly

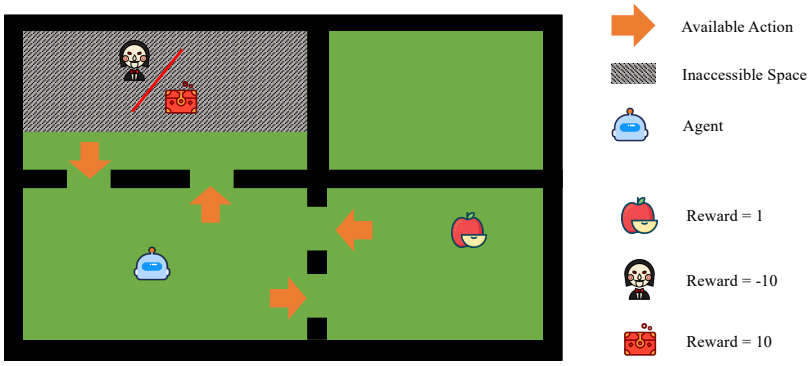
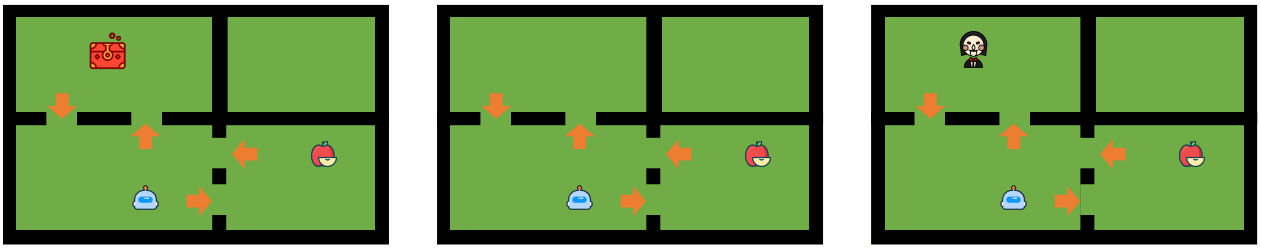


Fig. 3: Illustration of the Treasure-Explorer task. In this task, the policy needs to control an agent to move in the map. There are three reachable rooms in the environment (i.e., the lower-left room, the lower-right room, and the upper-left room). In each room, the agent can choose one of the available actions in the room (marked as the yellow arrows). By taking one of the available actions, the agent will move to the room pointed by the arrow. After moving into different rooms, the agent will hit the object in the room and get a reward, then get back to the lower-left room immediately. This is an environment with uncertainty. In the upper-left room, the object in the inaccessible space is unknown. That is, we do not know the next state by taking the "up" action in the lower-left room. The inaccessible space here is simplified by omitting the possibility of reaching the other two rooms to simplify the analysis.



(a) Treasure (b) Nothing (c) Ghost

Fig. 4: All possible dynamics models in the Treasure-Explorer task.

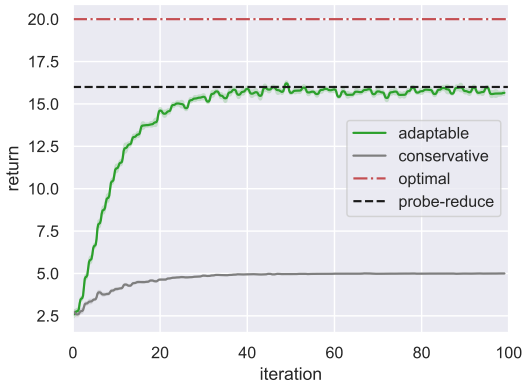


Fig. 5: The learning curves of the adaptable policy and conservative policy.

worse than the conservative policy. On the other hand, as in the case of "Treasure", the adaptable policy can reach a better performance than the conservative policy, with a larger performance gap of Δ_c.

6 Experiments

We evaluate MAPLE on multiple offline MuJoCo tasks [19]. All the details of MAPLE's training and evaluation are given in Appendix C and Appendix E. We release our code at Github *.

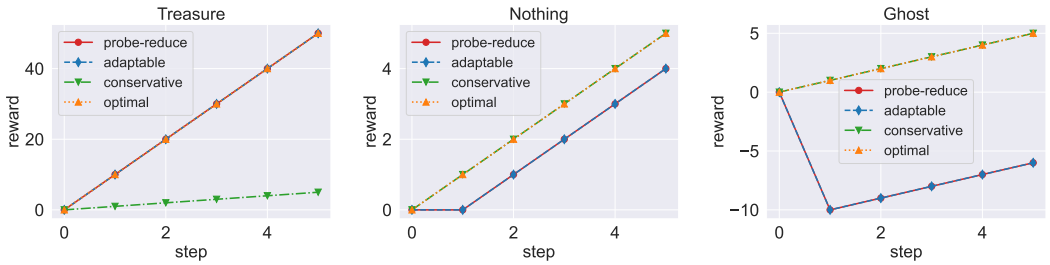
6.1 Comparative Evaluation

6.1.1 In D4RL Benchmark Tasks

We test MAPLE in standard offline RL tasks with D4RL datasets [45]. The version of the offline datasets we used is "v0". The ensemble dynamics model set is trained via supervised learning. We repeat each task with three random seeds. In the model learning stage, we train 20 models for each task and select 14 of them as the ensemble model for policy learning. The horizon H is set to 10 in these tasks. The policy is trained for 1000 iterations in the policy learning stage.

We compare MAPLE with: (1) MOPO: Learn an ensemble model via supervised learning and learn a policy in the ensemble models with uncertainty penalty [15]; (2) MOPO-loose: MOPO algorithm with the same hyperparameter as MAPLE for constraints, which is looser than MOPO; (3)

*. https://github.com/xionghuichen/MAPLE



(a) Treasure (b) Nothing (c) Ghost

Fig. 6: The up-to-now cumulative rewards at each step for adaptable, conservative, probe-reduce, and optimal policies.

TABLE 1: Results on D4RL benchmark [45]. Each number is the normalized score proposed by Fu et al. [45] of the policy at the last iteration of training, ± standard deviation. Among the offline RL methods, we bold the highest mean for each task.

Environment	Dataset	MAPLE	MOPO	MOPO-loose	SAC	BEAR	BC	BRAC-v	CQL
Walker2d	random	**21.7 ± 0.3**	13.6 ± 2.6	8.0 ± 5.4	4.1	6.7	9.8	0.5	7.0
Walker2d	medium	56.3 ± 10.6	11.8 ± 19.3	32.6 ± 18.0	0.9	33.2	6.6	**81.3**	79.2
Walker2d	mixed	**76.7 ± 3.8**	39.0 ± 9.6	35.7 ± 2.2	3.5	25.3	11.3	0.4	26.7
Walker2d	med-expert	73.8 ± 8.0	44.6 ± 12.9	66.7 ± 14.8	-0.1	26.0	6.4	66.6	**111.0**
HalfCheetah	random	**38.4 ± 1.3**	35.4 ± 1.5	35.4 ± 2.1	30.5	25.5	2.1	28.1	35.4
HalfCheetah	medium	**50.4 ± 1.9**	42.3 ± 1.6	44.0 ± 1.6	-4.3	38.6	36.1	45.5	44.4
HalfCheetah	mixed	**59.0 ± 0.6**	53.1 ± 2.0	36.9 ± 15.0	-2.4	36.2	38.4	45.9	46.2
HalfCheetah	med-expert	**63.5 ± 6.5**	63.3 ± 38.0	15.0 ± 6.0	1.8	51.7	35.8	45.3	62.4
Hopper	random	10.6 ± 0.1	11.7 ± 0.4	10.6 ± 0.6	11.3	9.5	1.6	**12.0**	10.8
Hopper	medium	21.1 ± 1.2	28.0 ± 12.4	16.9 ± 2.4	0.8	47.6	29.0	32.3	**58.0**
Hopper	mixed	**87.5 ± 10.8**	67.5 ± 24.7	83.1 ± 6.5	1.9	10.8	11.8	0.9	48.6
Hopper	med-expert	42.5 ± 4.1	23.7 ± 6.0	25.1 ± 1.8	1.6	4.0	111.9	0.8	**98.7**

BEAR: Learn a policy via off-policy RL while constraining the maximum mean discrepancy of the current policy to the behavior policy [14]; (4) BC: Imitate the behavior policy via supervised learning; (5) SAC: Perform typical SAC updates with the static dataset [42]; (6) BRAC-v: A behavior-regularized actor-critic proposed by Wu et al. [24]; (7) CQL: Learn an action-value function with regularization to obtain a conservative policy [46]. Results of (3-7) and (1) are taken from the work of [45] and [15].

Table 1 has shown the performance of 12 tasks. In summary, the performance of MAPLE on 7 tasks is better than other SOTA algorithms. Besides, MAPLE reaches the best performance among the SOTA model-based conservative policy learning algorithms in **10 out of the 12 tasks**. These results demonstrate the superior generalization ability of MAPLE.

However, in Hopper experiments, model-free methods BC, CQL, and BRAC-v often reach better performance. We consider that it is because the environment is unstable, more diverse collected data is needed for robust dynamics model learning. Even in MAPLE, a finite number of ensemble dynamics models might not cover the real case so that to make a robust adaptation. Therefore, only in the "Hopper-mixed" task, which has diverse collected data, MOPO and MAPLE can improve the deployment performance.

We also conduct MOPO-loose for each task, which shares the same hyper-parameters with MAPLE, including the ensemble model size, rollout length, and penalty coefficient. The results show that for some cases (e.g., Walker2d-med-expert and Hopper-mixed), MOPO-loose can also enhance the performance. We consider the improvement coming from the constraints in original implementations to be too tight. However, in most of the tasks, the improvement is not as significant as MAPLE. This phenomenon indicates that the improvement of MAPLE does not come from parameter tuning but our self-adaptation mechanism.

6.1.2 *In NeoRL Benchmark Tasks*

We also evaluate MAPLE in a recently proposed benchmark, NeoRL, a near real-world benchmark for offline reinforcement learning [47]. In particular, we select three MuJoCo tasks: HalfCheetah, Hopper and Walker2d which use the low and medium qualities of behavior policies for 1000-trajectory offline dataset collection to test the performance of MAPLE. In all of the dataset, we use $m = 50$. In HalfCheetah, we select $\lambda = 1.0$ and $H = 15$; In Walker2d, we select $\lambda = 0.25$ and $H = 15$; and in Hopper, we select $\lambda = 1.0$ and $H = 10$. The results are listed in Tab. 2. As can be seen in Tab. 2, 5 of the 6 tasks, MAPLE reaches the best performance among the offline model-based algorithms, while 2 of the 6 tasks, MAPLE reaches the best performance among the offline algorithms.

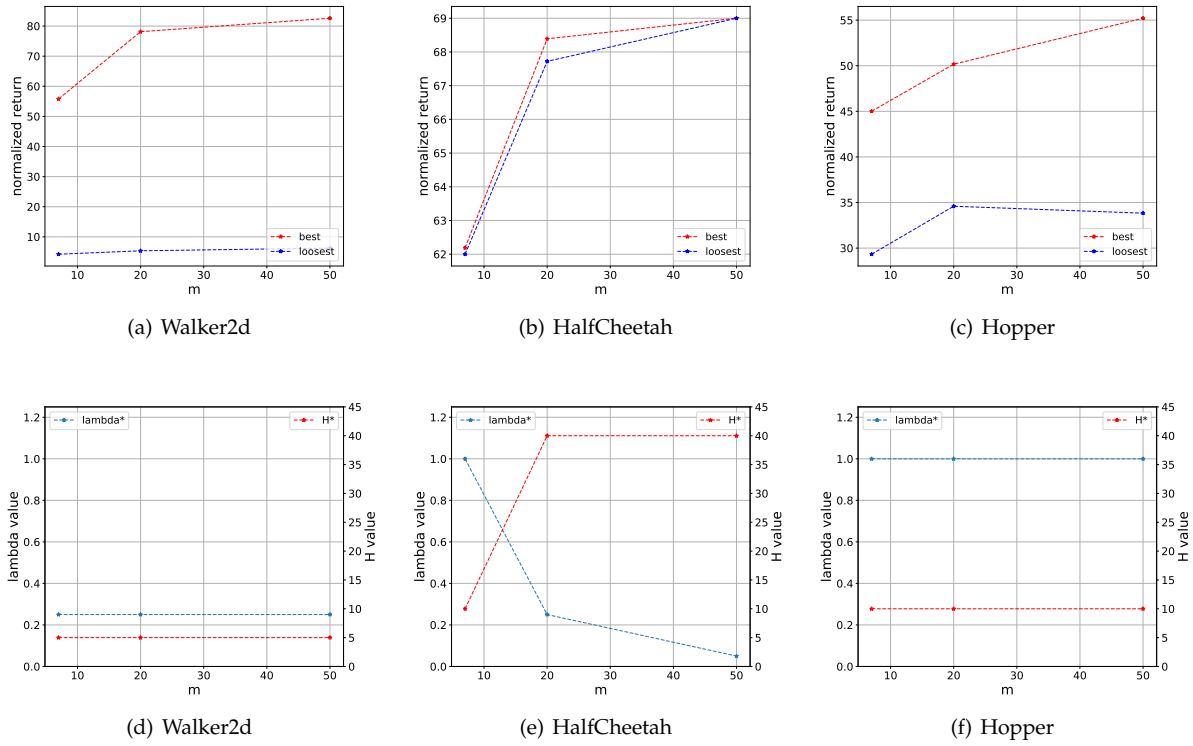
6.2 Hyper-parameters Analysis

We test the performance of MAPLE with different rollout length H, dynamics model size m, and penalty coefficient λ. We search over $H \in \{5, 10, 40\}$, $m \in \{7, 20, 50\}$ and $\lambda \in \{1.0, 0.25, 0.05\}$ in the task Walker-medium, Halfcheetah-mixed, and Hopper-medium-expert. For each setting, we run with three random seeds. We summarize the results in Table 8 and Figure 8. The learning curves of each setting can be found in Figure 16, Figure 17 and Figure 18.

First, as claimed in this work, tight constraints indeed lead to conservative policies, but the threshold starting to

TABLE 2: Results on NeoRL benchmark [47]. Each number is the raw score proposed by Qin et al. [47] of the policy at the 1000 iteration of training. We use $\pm$ to denote the standard deviation. Among all offline RL methods, we bold the highest mean for each task. Among the model-based offline RL methods (i.e., BREMEN, MOPO, and MAPLE), we mark the highest score with"*" for each task.

Environment	Dataset	MAPLE	MOPO	BC	BCQ	PLAS	CQL	CRR	BREMEN
Walker2d	low	1741.7 $\pm$ 221.5*	599.4 $\pm$ 725.3	1466.5 $\pm$ 99.8	1953.6 $\pm$ 231.6	2166.8 $\pm$ 531.1	**2298.8 $\pm$ 139.1**	1753.1 $\pm$ 90.4	1667.9 $\pm$ 449.1
Walker2d	medium	2095.9 $\pm$ 533.9*	2051.9 $\pm$ 104.6	2503.3 $\pm$ 97.9	**3173.7 $\pm$ 27.8**	1778.7 $\pm$ 679.5	2947.7 $\pm$ 49.7	2300.4 $\pm$ 354.3	1927.8 $\pm$ 852.2
HalfCheetah	low	3900.0 $\pm$ 20.8	**4741.1 $\pm$ 117.2***	3363.8 $\pm$ 27.0	3993.0 $\pm$ 49.9	3548.5 $\pm$ 3.8	4512.4 $\pm$ 65.5	3372.6 $\pm$ 24.4	4681.1 $\pm$ 230.9
HalfCheetah	medium	**8445.0 $\pm$ 69.8***	7534.7 $\pm$ 134.7	5866.8 $\pm$ 73.6	6062.8 $\pm$ 12.1	6092.3 $\pm$ 47.9	6576.2 $\pm$ 39.1	5137.8 $\pm$ 328.7	6666.9 $\pm$ 382.6
Hopper	low	**752.4 $\pm$ 39.8***	209.9 $\pm$ 101.1	502.6 $\pm$ 23.1	601.1 $\pm$ 6.3	638.9 $\pm$ 52.3	530.7 $\pm$ 4.0	557.1 $\pm$ 20.6	708.8 $\pm$ 250.7
Hopper	medium	917.6 $\pm$ 59.5*	39.0 $\pm$ 48.8	1692.0 $\pm$ 894.1	1573.6 $\pm$ 364.4	2018.2 $\pm$ 848.8	**2124.8 $\pm$ 231.8**	1576.2 $\pm$ 346.2	816.5 $\pm$ 181.7



(a) Walker2d (b) HalfCheetah (c) Hopper

(d) Walker2d (e) HalfCheetah (f) Hopper

Fig. 7: Illustration of hyper-parameters analysis on m. In the first row, we compare the normalized return of the best setting and the loosest setting. The x-axis is the model size m. For each m, the legend "best" is the setting that has the largest performance, among which model size is m. The legend "loosest" is the setting that $\lambda = 0.05$ and $H = 40$. In the second row, we compare the best constraint setting for each model size m. For each m, the legend "lambda*" is the setting that λ value of the best-performance setting among which model size is m. Similarly, the legend "H*" is the setting that H value of the best-performance setting among which model size is m.

suppress the asymptotic performance of policies is varied to the tasks. As can be seen in Figure 8(d), in the tasks of walker-medium, $\lambda = 1.0$ is the setting significantly suppressing the asymptotic performance, while in hopper-medium-expert task, as can be seen in Figure 8(f), $\lambda = 1.0$ is a good setting for policy learning. In hopper-medium-expert and halfcheetah-mixed tasks, $H = 5$ is the setting significantly suppressing the asymptotic performance (see Figure 8(c) and Figure 8(b)), while $H = 5$ and $H = 10$ are both good settings in walker-medium task (see Figure 8(b) and Figure 8(a)).

Second, as we do not construct all possible transitions in out-of-support regions, too loose constraints also hurt the asymptotic performance. In the task of walker2d-medium, $H = 40$ makes the asymptotic performance much worse than other settings. In the task of hopper-medium-expert, $\lambda = 0.05$ and $H = 40$ also degenerate the asymptotic performance. However, in the task of halfcheetah-mixed, the performance can always be improved by relaxing the constraints. The reason is probably that the environment of halfcheetah is more stable than other tasks. In the halfcheetah environment, trajectories will not pre-terminate no matter how badly the agent behaves. Therefore, the environment is tolerable to some incorrect actions. While in the rest tasks, the agent might reach unsafe states after performing some undesired actions. The incorrect actions might come from the probing process of MAPLE and errors of inference of the environment-context for some time-steps. Since the environment is tolerable to some incorrect actions, the probing processing and the generalization error of the environment-context extractor have less impact on the deployment performance.

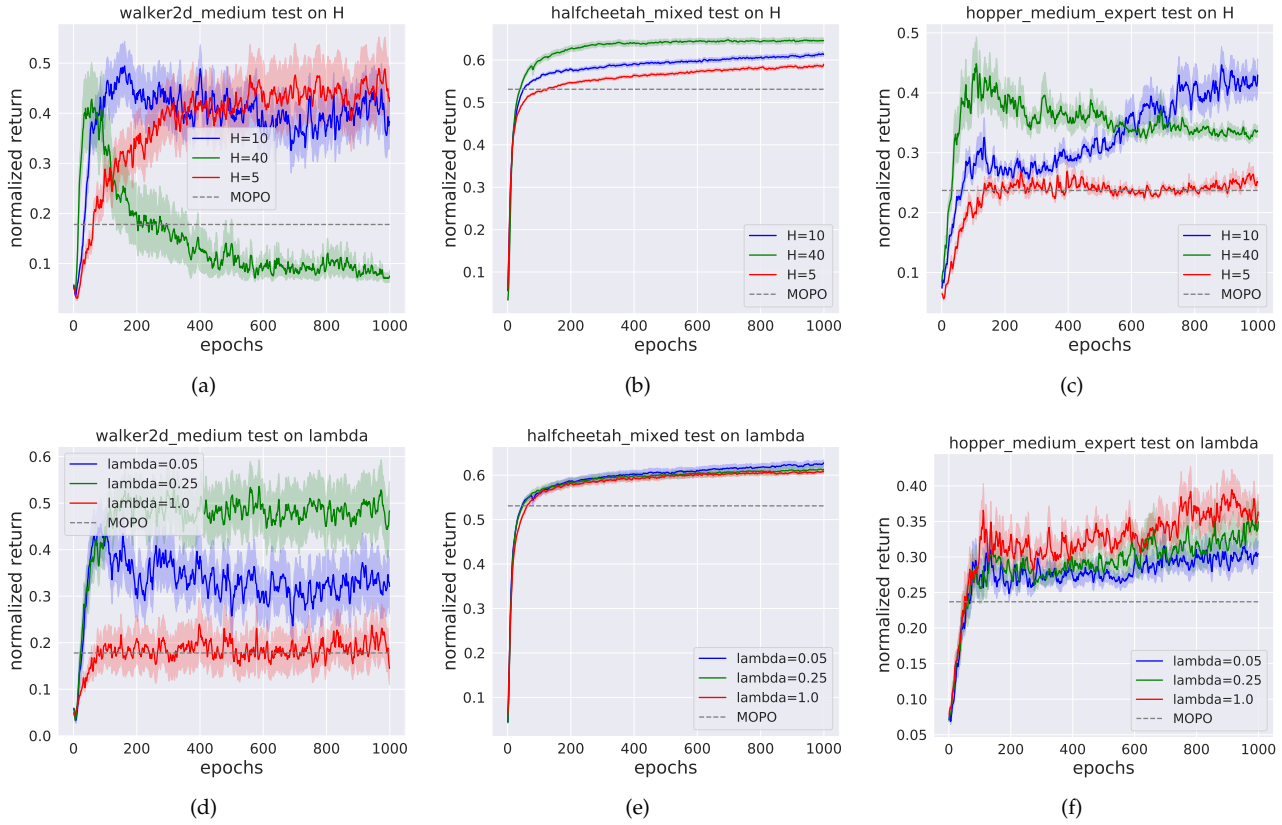


Fig. 8: Illustration of hyper-parameters analysis. In Figure 8(a), Figure 8(b) and Figure 8(c), we group the results by H. For each group, there have nine different experiments. Similarly, in Figure 8(d), Figure 8(e) and Figure 8(f), we group the results by m.

Third, for any type of constraint, increasing the size of the dynamics model set can improve the performance of MAPLE. Besides, as increasing enough size of the dynamics model set, the setting with loose constraints can reach better performance. The illustration can be found in Figure 7. As can be seen in Figure 7(a), Figure 7(b) and Figure 7(c), as m increased, the best-setting performances are significantly improved. Besides, as can be seen in Figure 7(e) and Figure 7(b), in the task of halfcheetah-mixed, when m increased, the best-setting is relaxed (i.e., $\lambda*$ is reduced and $H*$ is longer) and the performance of the loosest setting is close to the best setting gradually. However, in the task of hopper-medium-expert and walker-medium, the performances of the loosest settings are significantly worse than the best setting. The constraints of $H*$ and $\lambda*$ in the best setting keep the same from $m = 7$ to $m = 50$. We think the reason is that the size of dynamics models is not so large enough to cover the real cases in the out-of-support regions within the explorable boundary. Thus, the extractor ϕ can not infer a correct environment-context when deployed.

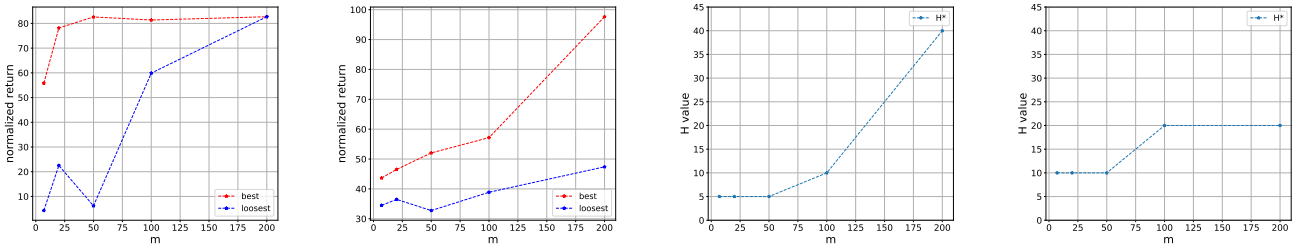
6.3 The Ability of Decision-Making in Out-of-Support Regions

The ultimate target of MAPLE is handling the decision-making challenge in out-of-support regions. By constructing all possible transitions in out-of-support regions, the probing-reducing loop can find the optimal policy finally. However, it is impractical to construct a dynamics model set that covers all possible transitions in out-of-support regions. In practice, we construct an ensemble dynamics model set and use loose constraints on policy sampling to expand the exploration boundary for better asymptotic performance. Therefore, we state that, with a large size of the model set, the dynamics models can cover more real transitions in out-of-support regions, then MAPLE is expected to make correct decisions in wider out-of-support regions. In the following, we verify the claim through three aspects:

- In Sec. 6.3.1, we select three tasks to show the effect of the scale of the dynamics model set to the out-of-support regions decision-making ability of MAPLE;
- In Sec. 6.3.2, we construct an extremely large dynamics model sets for each of the 9 tasks for MAPLE training to verify the statement further;
- As MAPLE trains the policy to adapt to any possible dynamics models in out-of-support regions, it has the potential to adapt to the environments that differ from where the offline data is gathered. In Sec. 6.3.3, we demonstrate the ability of MAPLE by deploying it in changed environments.

6.3.1 *MAPLE for Long-Horizon Decision-Making*

In this section, to verify the above statement, we first analyze the relationship among the rollout horizon H, the size of ensemble models m, and the asymptotic performance, where



(a) Walker2d (b) Hopper (c) Walker2d (d) Hopper

Fig. 9: Illustration of hyper-parameters analysis on m. In the first row, we compare the normalized return of the best setting and the loosest setting. The x-axis is the model size m. For each m, the legend "best" is the setting that has the largest performance, among which model size is m. The legend "loosest" is the setting that $H = 40$. In the second row, we compare the best constraint setting for each model size m. For each m, the legend "H*" is the setting that H value of the best-performance setting among which model size is m.

the rollout length represents the region of decision-making. If a policy trained with a longer H has better asymptotic performance, it has a better ability for decision-making in wider out-of-support regions. The asymptotic performance is evaluated by the cumulative rewards at the 1000-th iteration. We select $\lambda = 0.25$ for Walker2d-medium and $\lambda = 1.0$ for Hopper-medium-expert, and compare the performance of different H and m. We search over $H \in \{10, 20, 40\}$ and $m \in \{7, 50, 100, 200\}$ for conducting the experiments. The results can be found in Figure 20 in Appendix.

We merge that results and give an summary of the relation about asymptotic performance on different H and m in Figure 9 (inherited parts of the results in Sec. 6.2). In Figure 9, we can see that, by increasing m, the best setting is relaxed (i.e., $H*$ is longer) and the performance of the loosest setting is close to the best setting gradually. In particular, in Walker2d-medium, when $m = 200$, from $H = 10$ to $H = 40$, the normalized returns are all around 0.8, which also shows the robustness of MAPLE to different constraints with large model size. In Hopper-medium-expert, although there is still a gap between the performance of the best setting and the loosest setting, the performance of the loosest setting is gradually increased as m increased and the $H*$ is relaxed from 10 to 20.

On the other hand, we found that, without enough ensemble models, too loose constraints will result in worse performance. Taking walker2d as an example (see Table 3), for the setting of $m = 50$, when $H = 10$, the final normalized return is about 0.65. However, as H increases, the asymptotic performance drops gradually. When $H = 40$, the asymptotic performance is even worse than MOPO.

TABLE 3: Results of MAPLE in the Walker2d-medium task among different rollout lengths H and size of the ensemble model set m.

	H=10	$H = 20$	$H = 40$
$m = 50$	**65.59 ± 5.64**	41.22 ± 10.23	6.17 ± 0.37
$m = 200$	79.54 ± 0.25	80.23 ± 0.21	**83.46 ± 0.16**

We further verify the surmise through Figure 10. As can be seen in Figure 10, the trained one-step reward is similar among different horizons. It means that the performance of policies in dynamics models is similar. The worse performance indicates that the adaptable policy overfits the finite dynamics models and ϕ fails to infer a correct environment-context to the adaptable policy when deployed. The issue can be remedied by constructing more dynamics models.

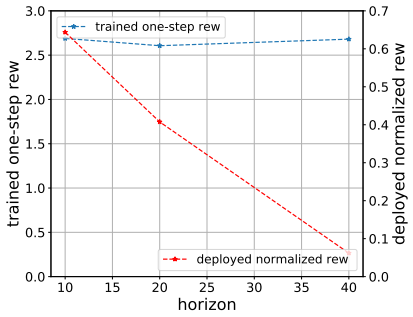


Fig. 10: A Comparison of trained and deployed rewards at the 1000-th epoch in the Walker2d-medium task based on $m = 50$ and among different H.

6.3.2 Long-Horizon Decision-Making of MAPLE with a Large Dynamics Model Set

Based on the above analysis, we can get an empirical conclusion that *increasing the model size is significantly helpful to find a better and robust adaptable policy via expanding the exploration boundary*. Therefore, we conduct another variant of the MAPLE algorithm, MAPLE-200, which uses 200 ensemble dynamics models for policy learning and expands the rollout horizon to 20.

The results of MAPLE-200 can be found in Table 4. We can see that the empirical conclusions not only suit the Walker2d-medium but also other tasks. In all of the tasks, MAPLE-200 reaches at least similar performance to MAPLE. In the tasks like Walker2d-med-expert, HalfCheetah-mixed, Hopper-medium, and Hopper-med-expert, the performance improvement of MAPLE-200 is significant. Besides, in the tasks of Hopper-medium and Hopper-med-expert, where MAPLE and MOPO fail to reach a comparable performance to model-free offline methods, MAPLE-200 can reach similar or much better results.

TABLE 4: Results on MuJoCo tasks with MAPLE-200. We overwrite the name of previous MAPLE algorithm with "MAPLE-20" for better clarity.

Environment	Dataset	MAPLE-200	MAPLE-20
Walker2d	random	**22.1** $\pm$ **0.1**	21.7 $\pm$ 0.3
Walker2d	medium	**81.3** $\pm$ **0.1**	56.3 $\pm$ 10.6
Walker2d	mixed	75.4 $\pm$ 0.9	**76.7** $\pm$ **3.8**
Walker2d	med-expert	**107.0** $\pm$ **0.8**	73.8 $\pm$ 8.0
HalfCheetah	random	**41.5** $\pm$ **3.6**	38.4 $\pm$ 1.3
HalfCheetah	medium	48.5 $\pm$ 1.4	**50.4** $\pm$ **1.9**
HalfCheetah	mixed	**69.5** $\pm$ **0.2**	59.0 $\pm$ 0.6
HalfCheetah	med-expert	55.4 $\pm$ 3.2	**63.5** $\pm$ **6.5**
Hopper	random	**10.7** $\pm$ **0.2**	10.6 $\pm$ 0.1
Hopper	medium	**44.1** $\pm$ **2.6**	21.1 $\pm$ 1.2
Hopper	mixed	85.0 $\pm$ 1.0	**87.5** $\pm$ **10.8**
Hopper	med-expert	**95.3** $\pm$ **7.3**	42.5 $\pm$ 4.1

MAPLE-200 demonstrates a powerful adaptation ability. However, we point out that, by increasing the 10x size of ensemble dynamics models, the time overhead for MAPLE-200 training is also larger. For example, by using NVIDIA Tesla P40 and Xeon(R) E5-2630 to train the algorithms, the time overhead of MAPLE-200 is 10 times longer than MAPLE. Besides, to obtain dynamics models that covered more real cases in out-of-support regions than MAPLE-200, the size of ensemble models becomes much larger, which is one of the limitations for current MAPLE implementation.

6.3.3 *Decision-Making Ability in Changed Environments*

As MAPLE trains the policy to adapt to any possible dynamics models in out-of-support regions, it has the potential to adapt to the environments that differ from where the offline data is gathered (we name it original environment). To verify this claim, we construct various environments by changing the dynamics parameters of the original environments. Specifically, we increase/reduce `density`, `gravity`, and `friction` by 50% in Walker2d, HalfCheetah, and Hopper, respectively. Then, we use the D4RL dataset [45] to train policies by several model-free and model-based offline RL methods. Finally, we evaluate the performance of the learned policies in the constructed environments.

The mean returns over at least three random seeds for each method are listed in Table 5. We can find that MAPLE-200 obtains the best returns in 4 out of 6 tasks and reaches the best averaged score, which implies the strong adaptability of MAPLE-200. We also find MAPLE-200 is inferior to MAPLE when reduce `gravity` by 50% in HalfCheetah-medium. We believe it is because of the hyper-parameters, which remain the same for all tasks and have not been fine-tuned for each task particularly. Moreover, the improvement of MAPLE-200 over MAPLE also shows the benefits of a large model set, which improves the adaptability a lot. The results in Table 5 strongly support the motivation of MAPLE, i.e. learning a adaptable policy that can adapt to all possible dynamics in out-of-support regions. Although MAPLE-200 only obtains the data in the original environments, its adaptability enables itself to survive in the environment that is different from the original environments.

6.4 MAPLE with other Meta-RL methods

In MAPLE, meta RL is adopted to solve the model-based offline RL problem. In the current implementation, the online system identification (OSI) method in meta RL is employed [30, 41, 32]. In fact, other techniques, e.g., VariBAD [48], can also be integrated into MAPLE. We also implemented VariBAD in MAPLE. We name the new combined method *VariBAD-MAPLE*. In VariBAD-MAPLE, we implement MAPLE with additional auxiliary tasks of the state and reward reconstruction and KL divergence minimization between the inferred z and a prior Gaussian distribution $\mathcal{N}(0, 1)$. All of the other techniques including the truncated horizon and reward penalty in MAPLE are also used in VariBAD-MAPLE. The comparison result is presented in Table 6. We name the MAPLE method instantiated with OSI-MAPLE to distinguish it from VariBAD-MAPLE.

As can be seen in VariBAD, just applying a meta-RL algorithm does not work well in the model-based offline RL domain compared with MAPLE. However, compared with the vanilla SAC algorithm, VariBAD can reach similar or better performance in most of the tasks. In the task of HalfCheetah-random, VariBAD can be even better than MOPO. However, by both considering the issues caused by the approximated dynamics models and the strategy of exploiting new samples obtained during deployment, VariBAD-MAPLE can also reach a significantly better performance than MOPO. Compared with the original implementation of OSI-MAPLE, VariBAD-MAPLE can do better than OSI-MAPLE in Walker2d-medium and HalfCheetah-random but worse than OSI-MAPLE in Walker2d-mixed and HalfCheetah-mixed.

The results again demonstrate the effectiveness of the pipeline of MAPLE. The results also inspire us that paying attention to the connection between the meta-RL techniques in sim2real and model-based offline-RL domains is valuable for model-based offline RL to develop more robust policy learning algorithms.

6.5 Visualization of the Hidden State

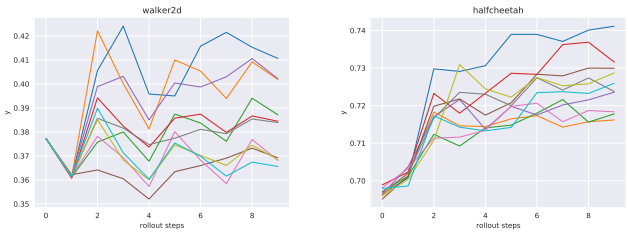


Fig. 11: Hidden state visualization in 10 learned dynamics models.

We conduct additional experiments in Walker2d-mixed and HalfCheetah-mixed to visualize the hidden state, z. We first randomly select $10,000$ states from the offline dataset and rollout the adaptable policy for 10 steps with each of the dynamics models in the ensemble model set. We record the absolute value of the context-variable $y_i = \frac{|z|_i}{|\text{Dim}(\mathcal{Z})|}$ for each step i, where $|\text{Dim}(\mathcal{Z})|$ denotes the number of dimensions of the embedding state. We present the results on 10 of the ensemble model in Figure 11. We found that y_i in different dynamics models are almost the same at the first step and become separable after $4-5$ steps. This result reveals that different dynamics models are distinguished by z. Besides, y

TABLE 5: Test results in changed environments. The policies learned by each method are evaluated in the dynamics different from where the offline data is gathered. The evaluation dynamics are constructed by changing the environment parameters, i.e., density, gravity, and friction. We bold the highest return for each task. We overwrite the name of previous MAPLE algorithm with "MAPLE-20" for clarity. We also implement "MOPO-200" which is a MOPO algorithm with 200 ensemble dynamics models. We overwrite the name of "MOPO" with "MOPO-20" for clarity.

Task	Env. change	MAPLE-200	MAPLE-20	MOPO-200	MOPO-20	CQL
Walker2d-medium	density × 50 %	**78.5**	5.5	0.0	49.4	10.8
Walker2d-medium	density× 150 %	**83.2**	5.6	0.0	9.4	16.2
HalfCheetah-medium	gravity × 50 %	30.0	36.0	39.5	**44.0**	21.3
HalfCheetah-medium	gravity × 150 %	**41.2**	32.2	38.8	33.6	31.7
Hopper-medium	friction × 50 %	**33.0**	12.0	12.2	15.6	1.2
Hopper-medium	friction × 150 %	21.1	13.2	**21.4**	12.6	1.4
Averaged score	/	**47.8**	17.4	18.6	27.4	13.7

TABLE 6: Comparisons on different variants of MAPLE in D4RL benchmark [45].

Environment	Dataset	OSI-MAPLE	VariBAD-MAPLE	VariBAD	SAC	MOPO
Walker2d	random	21.7 ± 0.3	**21.8 ± 0.2**	4.47 ±1.9	4.1	13.6 ±2.6
Walker2d	medium	56.3 ± 10.6	**81.1 ± 1.2**	4.5 ±0.6	0.9	11.8 ±19.3
Walker2d	mixed	**76.7 ± 3.8**	54.2 ± 8.7	10.8 ±2.4	3.5	39.0 ±9.6
Walker2d	med-expert	**73.8 ± 8.0**	70.0 ± 16.2	-0.1 ±0.0	-0.1	44.6 ±12.9
HalfCheetah	random	38.4 ± 1.3	**41.2 ±1.1**	37.8 ±0.2	30.5	35.4 ±1.5
HalfCheetah	medium	**50.4 ± 1.9**	**50.4 ±3.4**	22.4 ±3.7	-4.3	42.3 ±1.6
HalfCheetah	mixed	**59.0 ± 0.6**	56.7 ±0.5	42.2 ±5.7	-2.4	53.1 ±2.0
HalfCheetah	med-expert	63.5 ± 6.5	**64.9 ±6.4**	-0.2 ±0.5	1.8	63.3 ±38.0

are approximately divided into several groups. Hence, the context z is informative and has discovered the environment-specific information.

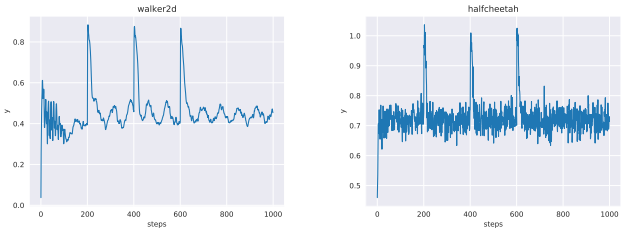


Fig. 12: Hidden state visualization in deployment environments. At time step 200, 400, and 600, the hidden state of the RNN is diturbed.

We also sample trajectories for 1000 time-step in the deployment environment. We found that the curves of y oscillate within a region. In HalfCheetah-mixed, the range is around $[0.6, 0.8]$. In Walker2d-mixed, the range is around $[0.3, 0.5]$. The z in the deployment environment are not constant, conversely, they are continuously changing. The changing range is approximately in the range of the converged value in the learned dynamics models. This result implies that the deployment environment could be a combination of the learned dynamics, and thus the context variables in the deployment environment could be all possible values that have appeared in the learned dynamics.

Finally, to further study the behavior of Φ in the deployment environment, we try to sample trajectories in the deployment environment and disturb the hidden state of the RNN at time step 200, 400, and 600. The context variables are robust to the disturbance. When the disturbance is injected, y_i will converge back to the region before the disturbance within 10 time steps. Thus, we also believe the predicted contexts are stable and robust.

7 Discussion and Future Work

Prior work has demonstrated the feasibility of model-based policy learning in the offline setting by using the conservative policy learning paradigm [15, 18]. It is an important breakthrough from zero to one in the offline setting, but it is far from the end of offline model-based policy learning. In this work, we investigate the decision-making problems in out-of-support regions directly. We first formulate the problem as decision-making in Pak-DM and propose MAPLE, a learn-to-adapt paradigm to solve the problem. We also give a theorem to describe the pros and cons of the paradigms to give us principles for the paradigm selection. We verified MAPLE in the MuJoCo tasks, and get our empirical conclusion: by increasing the size of the model set, we can expand the exploration boundary in the approximated dynamics models by using adaptable policy to make better and robust decisions in deployment environments.

MAPLE gives another direction to handle the offline model-based learning problem: Besides constraining on sampling and training dynamics models with better generalization, we can model out-of-distribution regions by constructing all possible transition patterns. The current limitation lies in the implementation: (1) The extractor's generalization ability depends on the neural network itself, which is uncontrollable to some degree. Adding some auxiliary tasks might handle this issue; (2) Ensemble is the direct way to cover real transitions in out-of-support regions. However, as the size of the model set becomes large, it is hard to generate new different dynamics models to cover the real cases only by increasing the size of the model. More efficient ways to increase the cover real transitions of the dynamics model set should be further explored.

ACKNOWLEDGEMENTS

This work is supported by National Key Research and Development Program of China (2020AAA0107200) and the National Science Foundation of China (61921006). We thank Zheng-Mao Zhu, Shengyi Jiang, Rong-Jun Qin, Yu-Ren Liu for their useful suggestions on the papers.

REFERENCES

[1] Xiting Wang, Yiru Chen, Jie Yang, Le Wu, Zhengtao Wu, and Xing Xie. A reinforcement learning framework for explainable recommendation. In *2018 IEEE International Conference on Data Mining*, pages 587–596. IEEE, 2018.

[2] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. Recommendations with negative feedback via pairwise deep reinforcement learning. In *Proceedings of the 24th. ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1040–1048, London, UK, 2018.

[3] Xiangyu Zhao, Changsheng Gu, Haoshenglun Zhang, Xiaobing Liu, Xiwang Yang, and Jiliang Tang. Deep reinforcement learning for online advertising in recommender systems. *CoRR*, abs/1909.03602, 2019.

[4] Han Cai, Kan Ren, Weinan Zhang, Kleanthis Malialis, Jun Wang, Yong Yu, and Defeng Guo. Real-time bidding by reinforcement learning in display advertising. In *Proceedings of the 10th. ACM International Conference on Web Search and Data Mining*, Cambridge, United Kingdom, 2017.

[5] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Edward Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *CoRR*, abs/2004.00784, 2020.

[6] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018.

[7] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[8] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Proceedings of the 2nd Annual Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 651–673. PMLR, 2018.

[9] Xinkun Nie, Emma Brunskill, and Stefan Wager. Learning when-to-treat policies. *CoRR*, abs/1905.09751, 2020.

[10] Adith Swaminathan, Akshay Krishnamurthy, Alekh Agarwal, Miroslav Dudík, John Langford, Damien Jose, and Imed Zitouni. Off-policy evaluation for slate recommendation. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3632–3642, Long Beach, CA, 2017.

[11] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *CoRR*, abs/2005.01643, 2020.

[12] Noah Y. Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin A. Riedmiller. Keep doing what worked: Behavior modelling priors for offline reinforcement learning. In *Proceeding of the 8th. International Conference on Learning Representations*, Addis Ababa, Ethiopia, 2020.

[13] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *Proceedings of the 36th. International Conference on Machine Learning*, Long Beach, CA, 2019.

[14] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*, Vancouver, Canada, 2019.

[15] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. MOPO: model-based offline policy optimization. *CoRR*, abs/2005.13239, 2020.

[16] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 12498–12509, 2019.

[17] Stéphane Ross and Drew Bagnell. Agnostic system identification for model-based reinforcement learning. In *Proceedings of the 29th International Conference on Machine Learning*, Edinburgh, UK, 2012.

[18] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. MOReL : Model-based offline reinforcement learning. *CoRR*, abs/2005.05951, 2020.

[19] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *Proceedings of the 24th. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, Vilamoura, Portugal, 2012.

[20] Alexandre Gilotte, Clément Calauzènes, Thomas Nedelec, Alexandre Abraham, and Simon Dollé. Offline A/B testing for recommender systems. In Yi Chang, Chengxiang Zhai, Yan Liu, and Yoelle Maarek, editors, *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 198–206. ACM, 2018.

[21] Georgios Theocharous, Philip S. Thomas, and Mohammad Ghavamzadeh. Personalized ad recommendation systems for life-time value optimization with guarantees. In *Proceedings of the 25th. International Joint Conference on Artificial Intelligence*, pages 1806–1812, Buenos Aires,

Argentina, 2015. AAAI Press.

[22] Philip S. Thomas, Georgios Theocharous, Mohammad Ghavamzadeh, Ishan Durugkar, and Emma Brunskill. Predictive off-policy policy evaluation for nonstationary decision problems, with applications to digital marketing. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the 31st. AAAI Conference on Artificial Intelligence*, pages 4740–4745, CA, USA, 2017. AAAI Press.

[23] Sascha Lange, Thomas Gabel, and Martin A. Riedmiller. Batch reinforcement learning. In Marco A. Wiering and Martijn van Otterlo, editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 45–73. Springer, 2012.

[24] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *CoRR*, abs/1911.11361, 2019.

[25] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. *International Conference on Machine Learning*, 2020.

[26] Chenjia Bai, Lingxiao Wang, Zhuoran Yang, Zhihong Deng, Animesh Garg, Peng Liu, and Zhaoran Wang. Pessimistic bootstrapping for uncertainty-driven offline reinforcement learning. *CoRR*, abs/2202.11566, 2022.

[27] Thanh Nguyen-Tang and Raman Arora. Viper: Provably efficient algorithm for offline RL with neural function approximation. In *Proceedings of the 11th International Conference on Learning Representations*, Kigali, Rwanda, 2023.

[28] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl2: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016.

[29] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th. International Conference on Machine Learning*, pages 1126–1135, Sydney, Australia, 2017.

[30] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real transfer of robotic control with dynamics randomization. In *Proceedings of the 35th. IEEE International Conference on Robotics and Automation*, pages 1–8, Brisbane, Australia, 2018.

[31] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *Proceedings of the 36th. International Conference on Machine Learning*, pages 5331–5340, Long Beach, CA, 2019.

[32] Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. In Nancy M. Amato, Siddhartha S. Srinivasa, Nora Ayanian, and Scott Kuindersma, editors, *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, , July 12-16, 2017*, Cambridge, MA, 2017.

[33] Wenxuan Zhou, Lerrel Pinto, and Abhinav Gupta. Environment probing interaction policies. In *7th International Conference on Learning Representations*, New Orleans, LA, 2019.

[34] Kimin Lee, Younggyo Seo, Seunghyun Lee, Honglak Lee, and Jinwoo Shin. Context-aware dynamics model for generalization in model-based reinforcement learning. *CoRR*, abs/2005.06800, 2020.

[35] Michael R. Zhang, Thomas Paine, Ofir Nachum, Cosmin Paduraru, George Tucker, Ziyu Wang, and Mohammad Norouzi. Autoregressive dynamics models for offline policy evaluation and optimization. In *Proceedings of the 9th International Conference on Learning Representations*, Virtual Event, 2021.

[36] Wenjie Shang, Yang Yu, Qingyang Li, Zhiwei Qin, Yiping Meng, and Jieping Ye. Environment reconstruction with hidden confounders for reinforcement learning based recommendation. In *Proceedings of the 25th. ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 566–576, Anchorage, AK, 2019.

[37] Xiong-Hui Chen, Yang Yu, Zheng-Mao Zhu, Zhihua Yu, Zhenjun Chen, Chenghe Wang, Yinan Wu, Hongqiu Wu, Rong-Jun Qin, Ruijin Ding, and Fangsheng Huang. Adversarial counterfactual environment model learning. *CoRR*, abs/2206.04890, 2022.

[38] Zheng-Mao Zhu, Xiong-Hui Chen, Hong-Long Tian, Kun Zhang, and Yang Yu. Offline reinforcement learning with causal structured world models. *CoRR*, abs/2206.01474, 2022.

[39] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[40] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063, Denver, Colorado, 1999.

[41] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019.

[42] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th. International Conference on Machine Learning*, pages 1856–1865, Stockholmsmässan, Sweden, 2018.

[43] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, 2014. Association for Computational Linguistics.

[44] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[45] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: datasets for deep data-driven reinforcement learning. *CoRR*, abs/2004.07219, 2020.

[46] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforce-

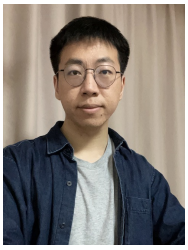
ment learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems*, 2020.

[47] Rongjun Qin, Songyi Gao, Xingyuan Zhang, Zhen Xu, Shengkai Huang, Zewen Li, Weinan Zhang, and Yang Yu. Neorl: A near real-world benchmark for offline reinforcement learning. *CoRR*, abs/2102.00714, 2021.

[48] Luisa M. Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: A very good method for bayes-adaptive deep RL via meta-learning. In *8th International Conference on Learning Representations, Addis Ababa, Ethiopia*, 2020.

[49] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

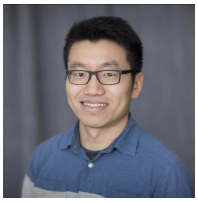
Xiong-Hui Chen received the B.Sc. degree from Southeast University, Nanjing, China, in 2018. Currently, he is working towards the Ph.D. degree in the National Key Lab for Novel Software Technology, the School of Artificial Intelligence, Nanjing University, China. His research focuses on handling the challenges of reinforcement learning in real-world applications. His works have been accepted in the top conferences of artificial intelligence, including NeurIPS, AAMAS, DAI, KDD, etc. He have served as a reviewer of NeurIPS, IJCAI, KDD, DAI, etc.



Fan-Ming Luo received the B.Sc. degree from Nanjing University, Nanjing, China, in 2019. Currently, he is working towards the Ph.D. degree in the National Key Lab for Novel Software Technology, the School of Artificial Intelligence, Nanjing University, China. His research interests include meta reinforcement learning and imitation learning. His work has been accepted in the top conferences of artificial intelligence, including NeurIPS and AAAI.

Yang Yu received his B.Sc and Ph.D. degrees in computer science from Nanjing University, China, in 2004 and 2011, respectively. Currently, he is a Professor in the School of Artificial Intelligence, Nanjing University, China. His research interest is in machine learning, and currently is mainly on real-world reinforcement learning. His work has been published in Artificial Intelligence, IJCAI, AAAI, NIPS, KDD, etc. He has was granted several conference best paper awards including IDEAL'16, GECCO'11, PAKDD'08, etc. He was granted CCF-IEEE CS Young Scientist Award in 2020, recognized as one of the AI's 10 to Watch by IEEE Intelligent Systems in 2018, and received the PAKDD Early Career Award in 2018. He was invited to give an Early Career Spotlight Talk in IJCAI'18.

Qingyang Li received his B.Sc degree from Beihang University, Beijing, China in 2012. He received his Ph.D. degree in computer science from Arizona State University, Arizona, United State in 2017. Currently, he is a research lead and manager at DiDi Labs, California, United State. His main research direction is reinforcement learning, deep learning and data mining. His work has been published in ICML, NeurIPS, KDD, WWW etc. He serves as the PC member and reviewers in the conferences of ICML, NeurIPS, ICLR, AAAI, TKDE and so on. He focused on cutting-edge research on reinforcement learning and deep learning, and promote the application and implementation of reinforcement learning in the field of real-world application.



Zhiwei Qin Zhiwei(Tony) Qin is Principal Scientist at Lyft Rideshare Labs, working on core problems in ridesharing marketplace optimization. Previously, he was Principal Research Scientist and Director of the Decision Intelligence group at DiDi AI Labs. Tony received his Ph.D. in Operations Research from Columbia University. His research interests span optimization and machine learning, with a particular focus in reinforcement learning and its applications in operational optimization, digital marketing, and smart transportation. He is Associate Editor of the ACM Journal on Autonomous Transportation Systems. He has published about 40 papers in top-tier conferences and journals in machine learning and optimization and served as Program Committee of NeurIPS, ICML, AAAI, IJCAI, KDD, and a referee of top journals including Transportation Research Part C and Transportation Science. He and his team received the INFORMS Daniel H. Wagner Prize for Excellence in Operations Research Practice in 2019 and were selected for the NeurIPS 2018 Best Demo Awards. Tony holds more than 10 US patents in intelligent transportation, supply chain, and recommendation systems.

Wenjie Shang Wenjie Shang received his B.Sc. and M.Sc degrees in computer science from Southeast University and Nanjing University, China, in 2016 and 2019, respectively. His research interests include machine learning, reinforcement learning, causal inference and autonomous driving. His work has been accepted in the top conferences and journals of artificial intelligence, including KDD, NeurIPS and MLj.

Jieping Ye received his Ph.D. degree in computer science from the University of Minnesota, Twin Cities, Minnesota, in 2005. He is currently a VP of Beike. He is also a professor at the University of Michigan, Ann Arbor, Michigan. His research interests include Big Data, machine learning, and data mining with applications in transportation and biomedicine. He won the NSF CAREER Award, in 2010 and the 2019 Daniel H. Wagner Prize for Excellence in the Practice of Advanced Analytics and Operations. His papers have been selected for the Outstanding Student Paper at ICML, in 2004, the KDD Best Research Paper Runner Up, in 2013, and the KDD Best Student Paper Award, in 2014. He has served as a senior program committee/area chair/program committee vice chair of many conferences, including NIPS, ICML, KDD, IJCAI, ICDM, and SDM. He has served as an associate editor for the Data Mining and Knowledge Discovery and the IEEE Transactions on Pattern Analysis and Machine Intelligence.

APPENDIX A
PROOF

Theorem 2. Given a target dynamics model ρ, a policy π_a learned by adapting, a policy π_c learned by constraints, and the maximum step N_m taken by π_a to probe and reduce the policy set to a single policy, the performance gap between π_a and π_c satisfies:

$$J_\rho(\pi_a) - J_\rho(\pi_c) \geq \Delta_c - \Delta_p - \gamma^{N_m+1} J_{\rho_\Delta}(\pi^*),$$

where Δ_c denotes the performance gap of the optimal policy π^* and π_c, while Δ_p denotes the performance degradation of MAPLE compared with π^* because of the phase of probing. $J_{\rho_\Delta}(\pi^*)$ denotes the performance degradation of π^* on the dynamics model ρ caused by different initial state distribution: $J_{\rho_\Delta}(\pi^*) = \mathbb{E}_{d^{\pi^\star}_{N_m+1}(s)}[V^\star(s)] - \mathbb{E}_{d^{\pi_a}_{N_m+1}(s)}[V^\star(s)]$, where $d^{\pi^\star}_{N_m+1}(s)$ and $d^{\pi_a}_{N_m+1}(s)$ denote the state distribution induced by $\pi^\star$ and π_a at the N_m+1 step and $V^*(s)$ denotes the expected long-term rewards of π^* at state s.

Proof A.1. MAPLE algorithm includes two policies: a probing policies π_p to visit state-action pair in inaccessible space, and the reduced policy π_t, which is equal to the optimal π^* in theory. Given a trajectory, the cumulative reward is:

$$\sum_{k=0}^{N} \gamma^k r^p_k + \sum_{k=N+1}^{T} \gamma^k r^*_k,$$

where N denotes the time-step for the probing policy π_p to reduce the policy set to the reduced policy π_t, which is equal to π^*. r^p and r^* denote the rewards run by π^p and π_t respectively. Regarding the policy of MAPLE as a mixed policy π_a, the performance can be written as:

$$J(\pi_a) = \mathbb{E}_{\tau\sim p(\tau|\pi_a,\rho)}\left[\sum_{k=0}^{N(\tau)} \gamma^k r^p_k + \sum_{k=N(\tau)+1}^{T} \gamma^k r^*_k\right],$$

where $N(\tau)$ denotes a function that outputs the time-step needed for the probing policy π_p to reduce the policy set for each trajectory τ. Assuming the maximized time-step needed is N_m, that is $N_m := \max_\tau N(\tau)$, we have:

$$\begin{aligned}
J(\pi_a) \geq& \mathbb{E}_{\tau\sim p(\tau|\pi_a,\rho)}\left[\sum_{k=0}^{N_m} \gamma^k r^p_k + \sum_{k=N_m+1}^{T} \gamma^k r^*_k\right] \\
=& \int \rho_0(s)\mathbb{E}_{s_0=s,\tau\sim p(\tau|\pi_p,\rho)}\left[\sum_{k=0}^{N_m} \gamma^k r^p_k\right] ds \\
&+ \gamma^{N_m+1}\int \rho_p(s)\mathbb{E}_{s_{N_m+1}=s,\tau\sim p(\tau|\pi^*,\rho)}\left[\sum_{k=0}^{T-N_m-1} \gamma^k r^*_{k+N_m+1}\right] ds \\
=& J^{\mathrm{par}}_\rho(\rho_0,\pi^p,N_m) + \gamma^{N_m+1} J^{\mathrm{par}}_\rho(\rho_p,\pi^*,T-N_m-1),
\end{aligned}$$

where $\rho_0(s)$ denotes the initial state distribution of the environments, and $\rho_p(s)$ denotes the state distribution at time-step N_m running with π_p. Here we introduce a new notation $J^{\mathrm{par}}_\rho(\rho_0,\pi,T)$ to describe the partial performance of policy π in dynamics model ρ, running with the initial state distribution ρ_0 and the horizon T. Then we know the performance gap between π^* and π_a:

$$\begin{aligned}
J(\pi^*) - J(\pi_a) \leq& J^{\mathrm{par}}_\rho(\rho_0,\pi^*,N_m) - J^{\mathrm{par}}_\rho(\rho_0,\pi^p,N_m) \\
&+ \gamma^{N_m+1} J^{\mathrm{par}}_\rho(\rho^* - \rho_p,\pi^*,T-N_m-1),
\end{aligned}$$

where $\rho^*(s)$ denote the state distribution at time-step N_m running with π^*. Assuming the performance gap between π^* and π_c is :

$$J(\pi^*) - J(\pi_c) = \Delta_c,$$

we have:

$$\begin{aligned}
J(\pi_a) - J(\pi_c) \geq& \Delta_c - \left(J^{\mathrm{par}}_\rho(\rho_0,\pi^*,N_m) - J^{\mathrm{par}}_\rho(\rho_0,\pi^p,N_m)\right) \\
&- \gamma^{N_m+1} J^{\mathrm{par}}_\rho(\rho^* - \rho_p,\pi^p,T-N_m-1).
\end{aligned}$$

$J^{\mathrm{par}}_\rho(\rho^* - \rho_p,\pi^*,T-N_m-1)$ is the performance degradation of π^* on the dynamics model ρ caused by different initial state distributions. In the infinite horizon setting, given an initial state s, $\sum_{k=0}^{T-N_m-1}\gamma^k r^*_{k+N_m+1} = V^*(s)$, where $V^*(s)$ denotes the expected long-term rewards of π^* taking state s as the initial state. Letting $d^{\pi^\star}_{N_m+1}(s)$ and $d^{\pi_a}_{N_m+1}(s)$ denote the state distribution induced by $\pi^\star$ and π_a at the N_m+1 step, we have:

$$\begin{aligned}
&J^{\mathrm{par}}_\rho(\rho^* - \rho_p,\pi^*,T-N_m-1) \\
=& \mathbb{E}_{d^{\pi^\star}_{N_m+1}(s)}[V^\star(s)] - \mathbb{E}_{d^{\pi_a}_{N_m+1}(s)}[V^\star(s)].
\end{aligned}$$

Letting $\Delta_p := J^{\mathrm{par}}_\rho(\rho_0,\pi^*,N_m) - J^{\mathrm{par}}_\rho(\rho_0,\pi^p,N_m)$, which is the performance degradation of MAPLE compared with π^* in the phase of probing in inaccessible space, and $J_{\rho_\Delta}(\pi^*) := J^{\mathrm{par}}_\rho(\rho_\Delta,\pi^*,T-N_m-1)$, which is the performance of π^* on the dynamics model ρ with an occupancy measure gap of the state distribution $\rho_\Delta = \rho^* - \rho_p$ at time-step N_m, we get our conclusion:

$$J_\rho(\pi_a) - J_\rho(\pi_c) \geq \Delta_c - \Delta_p - \gamma^{N_m+1} J_{\rho_\Delta}(\pi^*).$$

APPENDIX B
MAPLE IN IMPLEMENTATION

B.1 Network Structure

The network structure of MAPLE is shown in Figure 13. In the implementation, we use two independent neural network parameters for the Q-value function and policy. In Figure 13, we use the superscript of v and π for the parameters φ and ϕ to denote the independent neural network parameters for the Q-value function and policy. In MAPLE, there is a skip-connection for the environment-aware layer to reuse the original input features. The environment-context extractor layers are both modeled with a single-layer GRU cell [49]. Here we use h_t^v and h_t^π to denote the output of the two environment-context extractors. After the environment-context extractor layers, we use another fully-connected embedding layer f to reduce the output of the hidden context. Empirically, the output of the fully-connected embedding z should not be much larger than the dimension of x_t. Too large of z might lead to unstable policy and Q-value training. The embedding layer and environment-aware layer are modeled with Multilayer Perceptron (MLP). Table 7 reports the detailed parameters of the neural network. The hyper-parameters have not been fine-tuned, other structures can be tried: e.g., we found that the performance would be better if increasing the fully-connected embedding layers to 128 at least in HalfCheetah tasks.

B.2 Implementation Details

B.2.1 Reward penalty with uncertainty quantification

We use reward penalty and trajectory truncation as in MOPO [15], but the coefficients are more relaxed (i.e., smaller reward penalty coefficient and longer rollout length). We model the dynamics models with Gaussian distribution. For each time-step t, the reward penalty $U(s_t, a_t) = \max_i ||\Sigma^i(s_t, a_t)||_2$, where $\Sigma^i(s_t, a_t)$ denotes the standard deviation of the predicted Gaussian distribution of the i-th dynamics model at state s_t and action a_t, and $||\cdot||_2$ denotes the l2-norm.

B.2.2 State/penalty Clipping

As we increase the rollout length, the large compounding error might lead the agent to reach entirely unreal regions in which the states would never appear in the deployment environment. These samples are useless for adaptable policy learning and might make the training process unstable. Thus, we constrain the range of state to [-100, 100], and we also truncate trajectories when the predicted next states are out of range of [-100, 100]. For the same reason, the reward penalty is clipped to [0, 40].

B.2.3 Hidden State of RNN

As seen in Algorithm 1, when sampling initial states for the model-policy interaction, the hidden states are also sampled. The policy will also start inferring action starting from the hidden state. Thus, we should update the hidden states of the offline dataset $\mathcal{D}_{\text{off}}$ periodically. In our implementation, for every four epoch, we infer the hidden states of policy and value function for the dataset.

B.3 Hyper-Parameters

Other hyper-parameters include: rollout length H, dynamics model size m, and penalty coefficient λ. For all of the tasks, we use $H = 10$, $m = 20$ and penalty coefficient $\lambda = 0.25$ except $\lambda = 5.0$ in HalfCheetah-med-expert and $\lambda = 1.0$ and $H = 5$ in Hopper-mixed. The other hyper-parameters are the same as the original MOPO.

APPENDIX C
DETAILED EXPERIMENTAL SETTING IN MUJOCO TASKS

We test the algorithms in standard offline RL tasks with D4RL datasets [45]. In particular, we use data from 3 environments: Hopper, HalfCheetah, and Walker2d. In each environment, we test MAPLE with 4 four kinds of datasets: random, medium, mixed, and medium expert (med-expert). The datasets are gathered through different strategies: random and medium are collected by a random and medium policy, while med-expert is the concatenation of medium and the data collected by an expert policy. Mixed uses the replay buffer of a policy trained up to the performance of the medium agent. We repeat each task with three random seeds. In the model learning stage, we train 20 models for each task and select 14 of them as the ensemble model for policy learning. The ensemble dynamics model set is trained via supervised learning. The policy is trained for 1000 iterations.

APPENDIX D
EXTRA EXPERIMENTAL RESULTS IN THE TREASURE-EXPLORER TASK

In the Treasure-Explorer task, we present the trajectory sampled by the optimal policy, conservative policy, and adaptable policy in Figure 15. In the Nothing and the Ghost environments, the adaptable policy takes an active probing behavior at the first step. Then, it repeats the optimal policy for the rest steps. However, the conservative policy keeps a single behavior pattern: always picks the apple in the room on the right side. The trajectories match the behaviors of the probe-reduce policy: probing first and then exploiting accordingly.

APPENDIX E
EXTRA EXPERIMENTAL RESULTS IN MUJOCO TASKS

E.1 Learning Curves of MAPLE and MOPO-loose

The learning curves of MAPLE and MOPO-loose in 12 tasks can be found in Figure 14. MOPO-loose shares the same hyper-parameters with MAPLE, including the ensemble model size, the weights of parameters of each dynamics model, rollout length, and penalty coefficient. The results show that for some cases (e.g., Walker2d-med-expert and Hopper-mixed), MOPO-loose can also enhance the performance. We consider the improvement coming from the constraints in original implementations being too tight. However, in most of the tasks, the improvement is not as significant as MAPLE. This phenomenon indicates that the improvement of MAPLE does not come from parameter tuning but our self-adaptation mechanism.

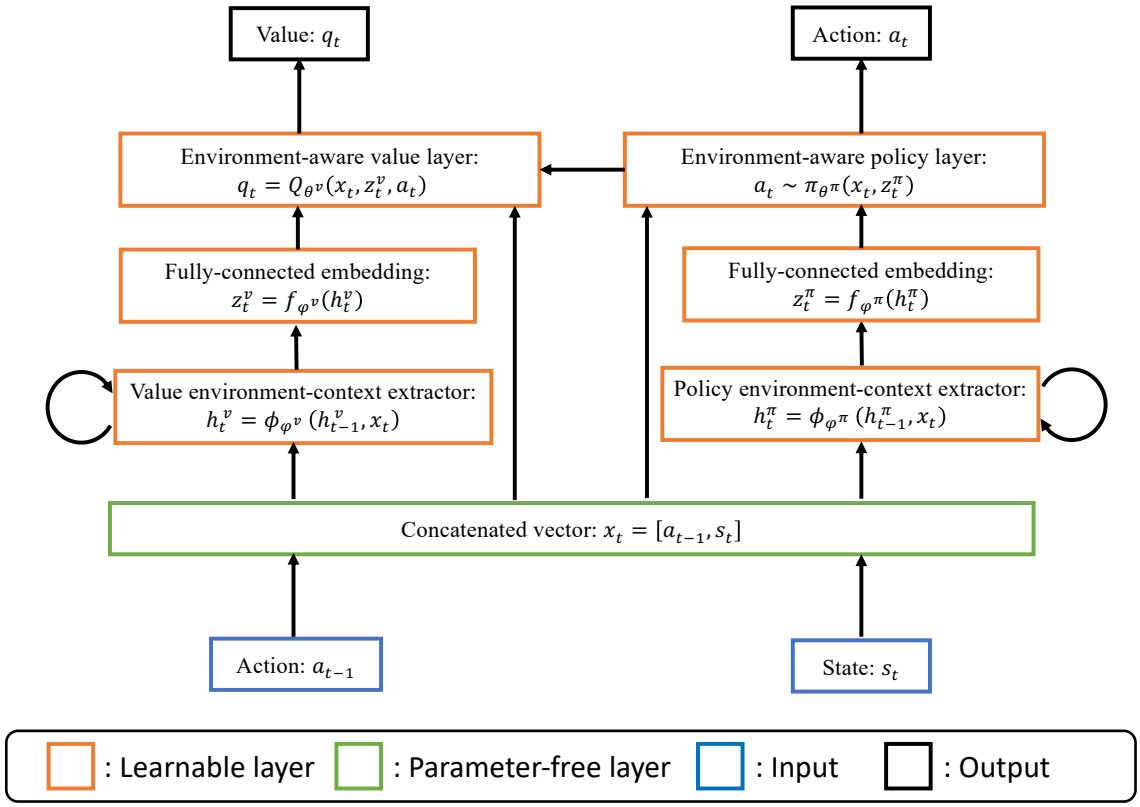


Fig. 13: Illustration of the network structure for MAPLE.

TABLE 7: The hyper-parameters of MAPLE for the neural network.

Hyperparameter	Value
Activation function of hidden layers	relu
Activation function of policy output	tanh
Activation function of q-value output	linear
Fully-connected embedding layers f	[16]
Unit of the environment-context extractor ϕ	128
Environment-aware layers of Q and π	[256, 256]

E.2 Learning Curves of MAPLE-200

Based on the above analysis, we can get an empirical conclusion that increasing the model size is significantly helpful to find a better and robust adaptable policy via expanding the exploration boundary. Therefore, we conduct another variant of the MAPLE algorithm, MAPLE-200, which uses 200 ensemble dynamics models for policy learning and expands the rollout horizon to 20. We select $\lambda = 0.25$ for tasks in HalfCheetah and Walker2d, and $\lambda = 1.0$ for Hopper. we still select $\lambda = 5.0$ and $H = 10$ in the task of HalfCheetah-med-expert, and we select $\lambda = 0.25$ and $H = 10$ in the task Hopper-mixed as before. We test MAPLE-200 in all of the tasks. The results can be found in Figure 19. The results of hyper-parameters analysis with a larger size of dynamics models is in Figure 20.

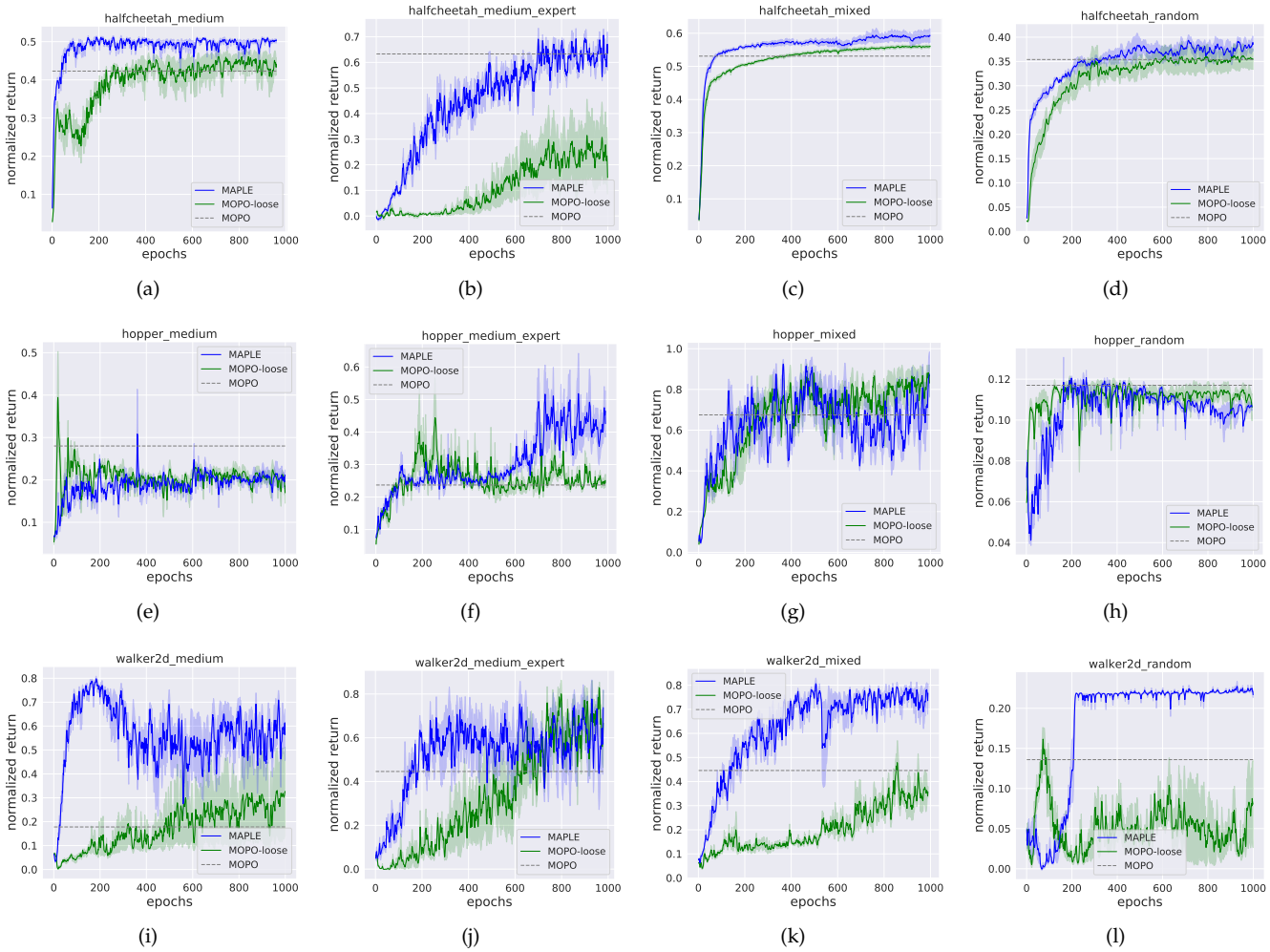
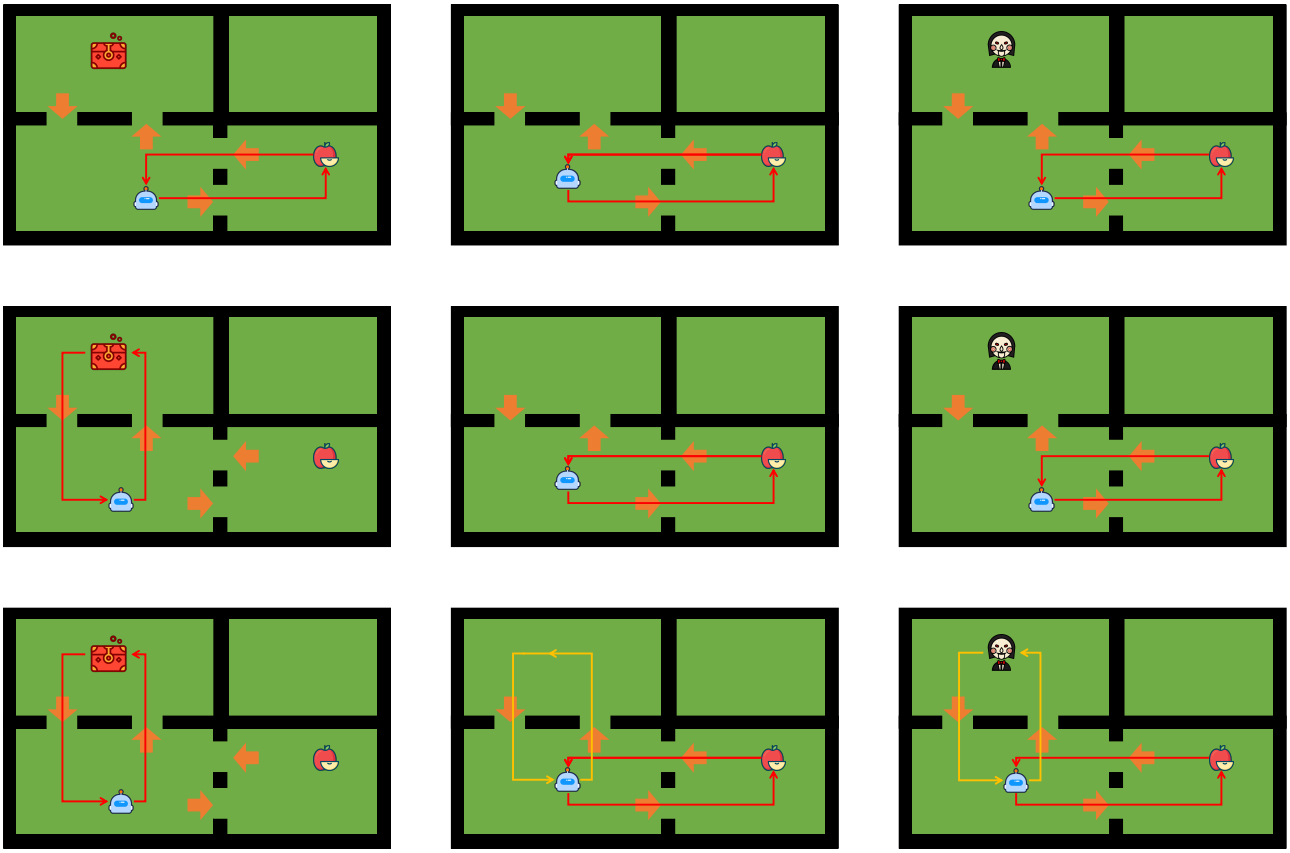


Fig. 14: The learning curves of MAPLE, MOPO, and MOPO-loose in mujoco tasks. The solid curves are the mean reward and the shadow is the standard error of three seeds.



(a) Treasure (b) Nothing (c) Ghost

Fig. 15: Trajectories of the conservative, adaptable, and optimal policies in each environment. The first row is the conservative policy; the second is the optimal policy; the third is the adaptable policy. In the Nothing and the Ghost environments, a yellow line indicates the probing behavior. The agent will execute the probing behavior for one time, and then continuously perform the red line behavior. In the Treasure environment, the probing behavior is the same as the optimal behavior.

TABLE 8: Results on D4RL benchmarks. Each number is the normalized score proposed by Fu et al. [45] of the policy at the last iteration of training, $\pm$ standard deviation. We group the experiments by the type of task and the value of m and bold the top-3 scores for each group.

setting			performance		
m	H	λ	walker-medium	halfcheetah-mixed	hopper-medium-expert
7	5	1.0	31.86 $\pm$ 20.22	59.69 $\pm$ 1.55	22.89 $\pm$ 0.06
		0.25	**55.81 $\pm$ 8.73**	57.75 $\pm$ 1.59	25.58 $\pm$ 2.12
		0.05	31.97 $\pm$ 7.61	58.51 $\pm$ 1.48	24.97 $\pm$ 5.25
	10	1.0	8.69 $\pm$ 0.26	**62.19 $\pm$ 0.24**	**45.01 $\pm$ 5.18**
		0.25	**37.70 $\pm$ 16.82**	58.84 $\pm$ 2.05	**43.59 $\pm$ 13.20**
		0.05	**41.50 $\pm$ 13.13**	**61.83 $\pm$ 1.58**	27.02 $\pm$ 0.75
	40	1.0	7.80 $\pm$ 0.44	60.80 $\pm$ 0.73	**38.11 $\pm$ 1.94**
		0.25	6.11 $\pm$ 0.39	61.59 $\pm$ 1.56	26.95 $\pm$ 0.80
		0.05	4.23 $\pm$ 1.80	**62.00 $\pm$ 1.55**	29.33 $\pm$ 0.19
20	5	1.0	5.37 $\pm$ 2.23	58.10 $\pm$ 1.06	24.92 $\pm$ 1.22
		0.25	**78.14 $\pm$ 1.12**	59.06 $\pm$ 0.53	24.99 $\pm$ 0.80
		0.05	**76.26 $\pm$ 0.97**	59.45 $\pm$ 0.56	24.82 $\pm$ 1.30
	10	1.0	33.37 $\pm$ 16.61	60.86 $\pm$ 0.65	**49.89 $\pm$ 3.83**
		0.25	**56.30 $\pm$ 10.60**	59.04 $\pm$ 0.60	**41.93 $\pm$ 4.93**
		0.05	14.91 $\pm$ 7.06	**63.18 $\pm$ 1.85**	**43.51 $\pm$ 5.86**
	40	1.0	30.50 $\pm$ 15.56	62.05 $\pm$ 0.17	33.98 $\pm$ 0.10
		0.25	22.53 $\pm$ 8.35	**68.39 $\pm$ 0.27**	36.48 $\pm$ 2.09
		0.05	5.36 $\pm$ 1.33	**67.72 $\pm$ 0.36**	34.59 $\pm$ 0.31
50	5	1.0	12.98 $\pm$ 1.29	58.01 $\pm$ 0.16	28.15 $\pm$ 2.29
		0.25	**82.61 $\pm$ 1.38**	59.24 $\pm$ 0.92	26.23 $\pm$ 0.16
		0.05	**54.70 $\pm$ 16.65**	60.79 $\pm$ 1.20	23.45 $\pm$ 0.63
	10	1.0	41.20 $\pm$ 18.24	61.94 $\pm$ 0.27	**55.21 $\pm$ 8.42**
		0.25	**79.35 $\pm$ 0.64**	59.14 $\pm$ 0.89	**46.52 $\pm$ 6.89**
		0.05	53.79 $\pm$ 8.42	**63.88 $\pm$ 1.39**	34.88 $\pm$ 3.59
	40	1.0	8.60 $\pm$ 0.03	61.27 $\pm$ 1.38	**38.55 $\pm$ 0.45**
		0.25	6.17 $\pm$ 0.37	**67.66 $\pm$ 0.66**	30.39 $\pm$ 0.16
		0.05	6.52 $\pm$ 0.09	**69.00 $\pm$ 0.21**	33.83 $\pm$ 2.91
MOPO			17.8 $\pm$ 19.3	53.1 $\pm$ 2.0	23.7 $\pm$ 6.0

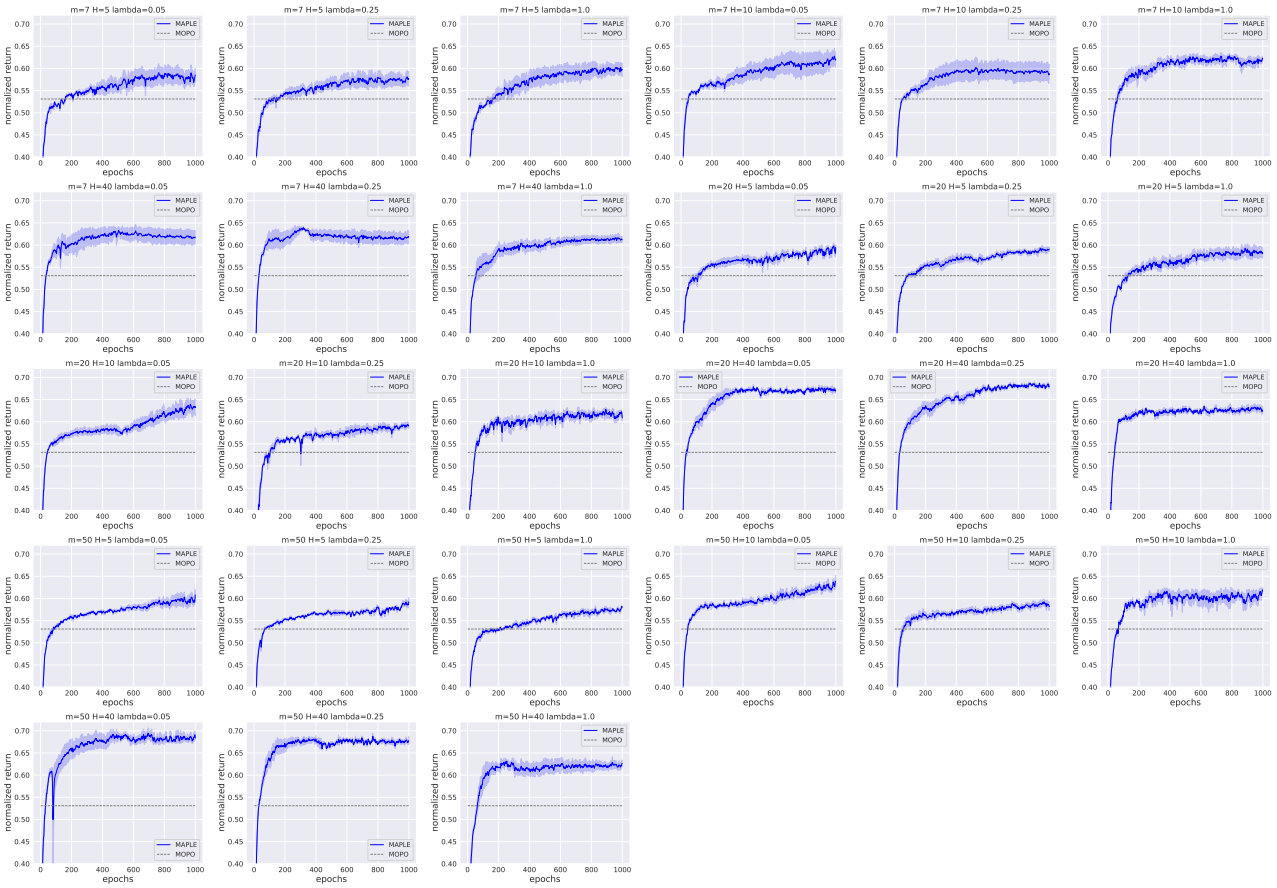


Fig. 16: Illustration of hyper-parameter analysis on Halfcheetah-mixed. The solid curves are the mean reward and the shadow is the standard error of three seeds.

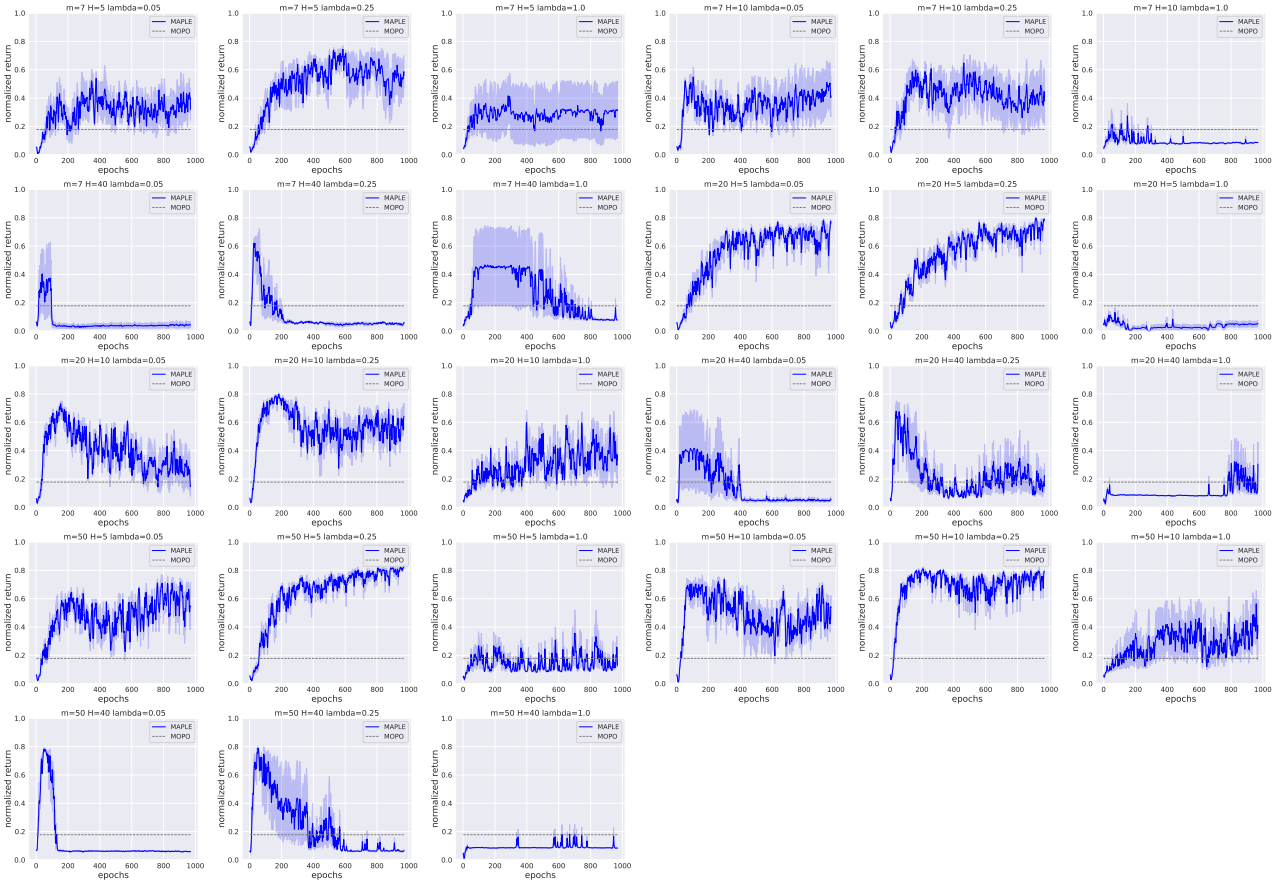


Fig. 17: Illustration of hyper-parameter analysis on Walker2d-medium. The solid curves are the mean reward and the shadow is the standard error of three seeds.

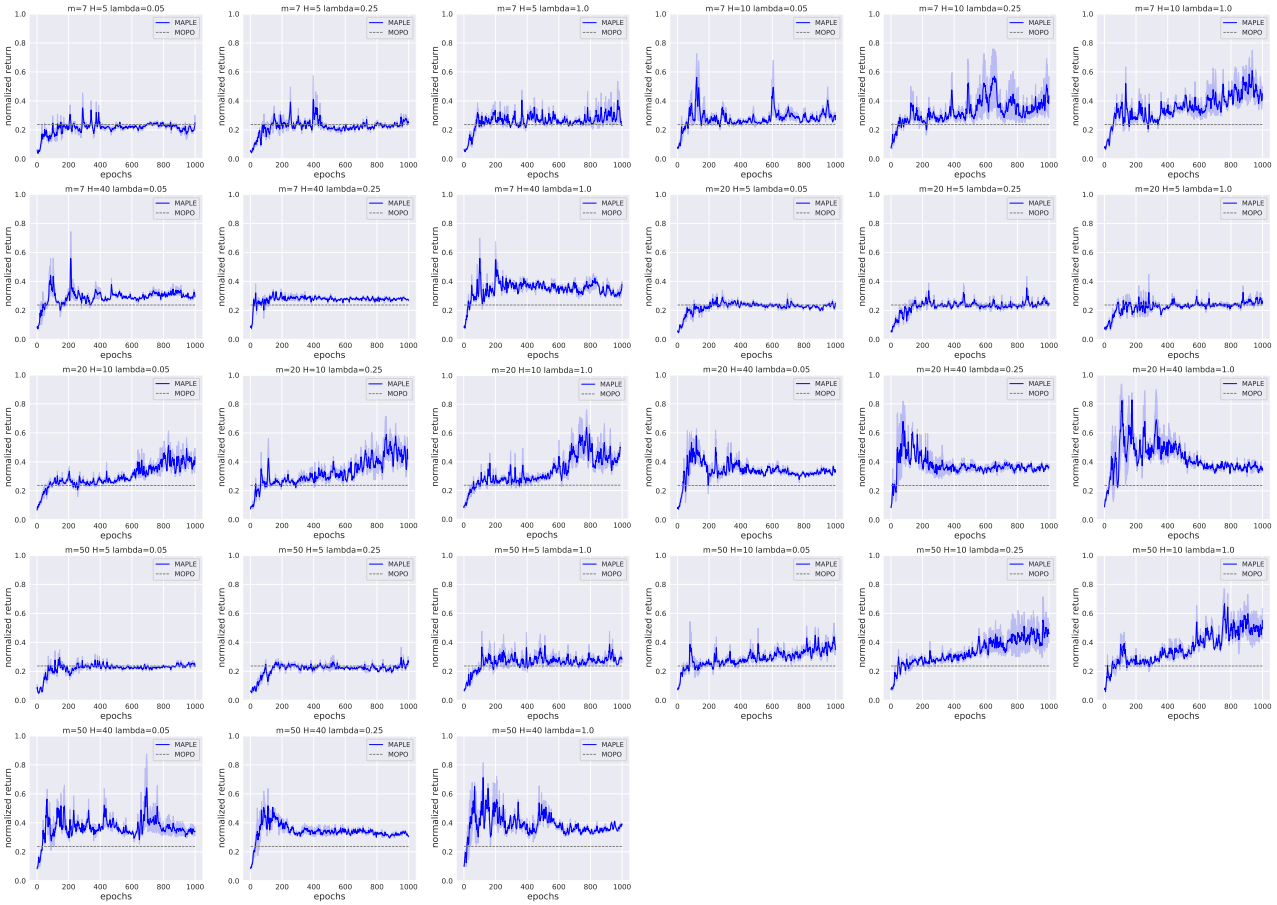


Fig. 18: Illustration of hyper-parameter analysis on Hopper-medium-expert. The solid curves are the mean reward and the shadow is the standard error of three seeds.

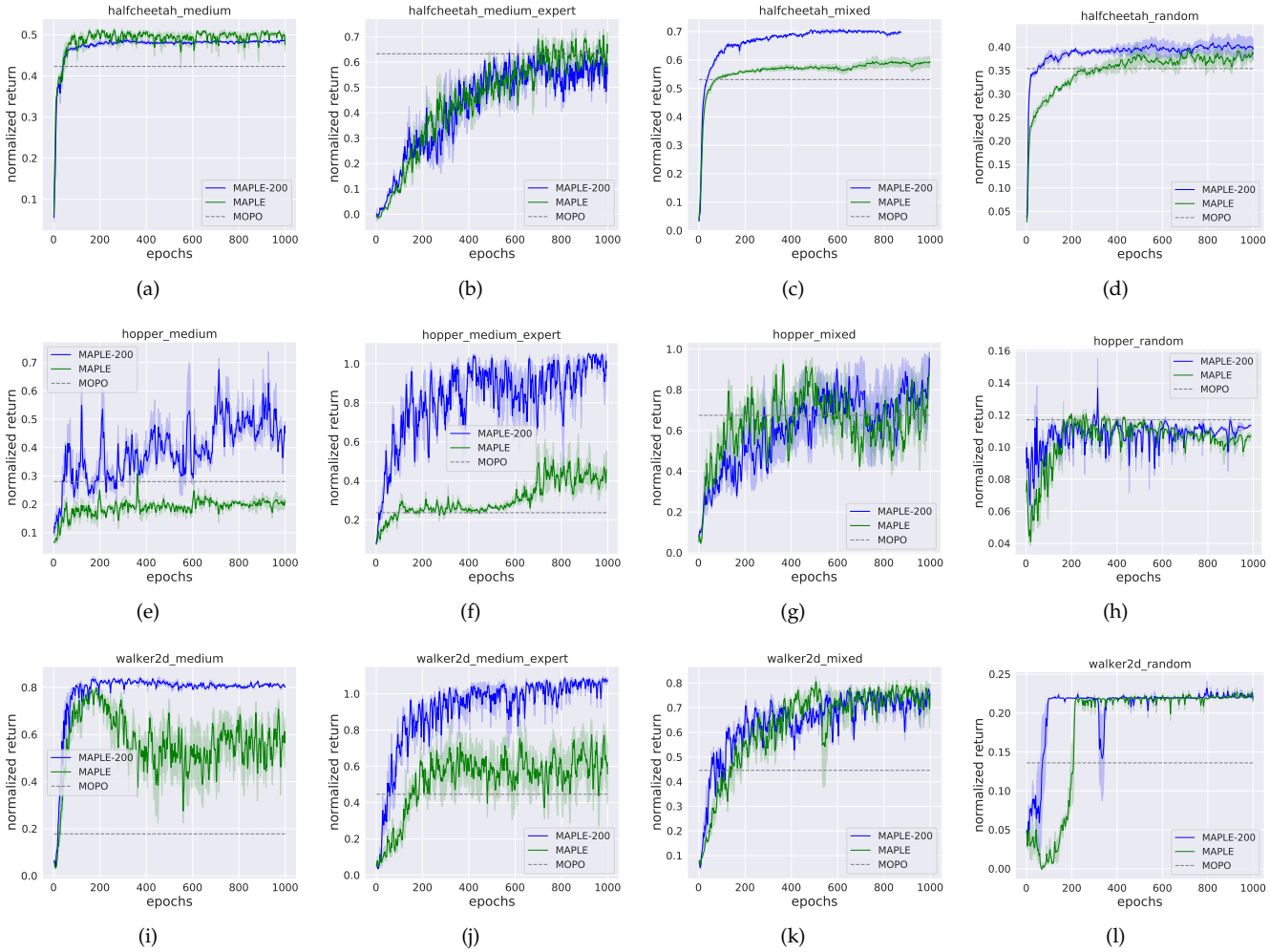
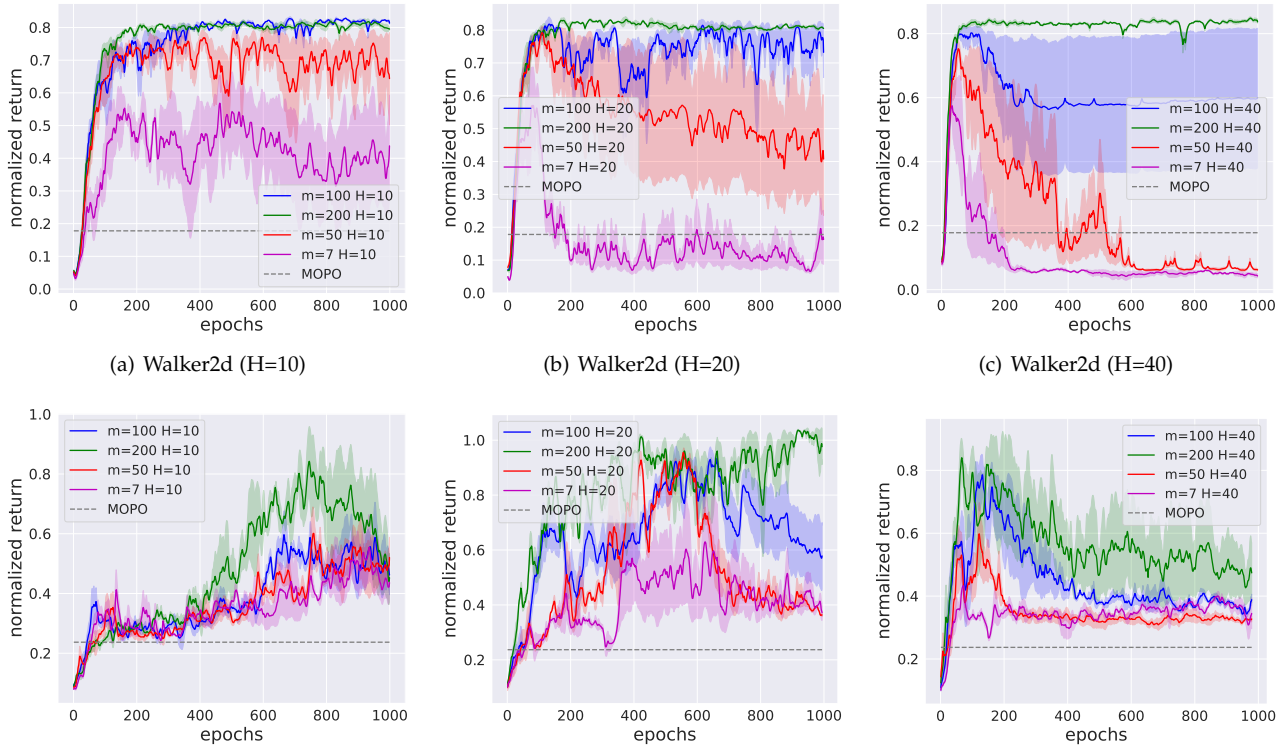


Fig. 19: Learning curves of MAPLE-200, MAPLE and MOPO in mujoco tasks. The solid curves are the mean reward and the shadow is the standard error of three seeds.



(a) Walker2d (H=10) (b) Walker2d (H=20) (c) Walker2d (H=40)

(d) Hopper (H=10) (e) Hopper (H=20) (f) Hopper (H=40)

Fig. 20: The learning curves of MAPLE with different hyper-parameters m and H. The solid curves are the mean of normalized return and the shadow is the standard error.