Journal of Artificial Intelligence Research 81 (2024) 579-617 Submitted 04/2024; published 11/2024

Efficiently Adapt to New Dynamic via Meta-Model

Kaixin Huang HKX981029@GMAIL.COM
Tsinghua Shenzhen International Graduate School, Shenzhen, Guangdong, China

Chen Zhao ZHAO-C21@MAILS.TSINGHUA.EDU.CN
Tsinghua Shenzhen International Graduate School, Shenzhen, Guangdong, China

Chun Yuan YUANC@SZ.TSINGHUA.EDU.CN
Tsinghua Shenzhen International Graduate School, Shenzhen, Guangdong, China

Abstract

We delve into the realm of offline meta-reinforcement learning (OMRL), a practical paradigm in the field of reinforcement learning that leverages offline data to adapt to new tasks. While prior approaches have not explored the utilization of context-based dynamical models to tackle OMRL problems, our research endeavors to fill this gap. Our investigation uncovers shortcomings in existing context-based methods, primarily related to distribution shifts during offline learning and challenges in establishing stable task representations. To address these issues, we formulate the problem as Hidden-Parameter MDPs and propose a framework for effective model adaptation using meta-models plus latent variables, which is inferred by the transformer-based system recognition module trained in an unsupervised fashion. Through extensive experimentation encompassing diverse simulated robotics and control tasks, we validate the efficacy of our approach and demonstrate its superior generalization ability compared to existing schemes, and explore multiple strategies for obtaining policies with personalized models. Our method achieves a model with reduced prediction error, outperforming previous methods in policy performance, and facilitating efficient adaptation when compared to prior dynamic model generalization methods and OMRL algorithms.

1. Introduction

Reinforcement learning (RL) has shown great success in solving sequential decision problems such as board games (Schrittwieser et al., 2020), computer games (Kaiser et al., 2019), and robotics (Zhang et al., 2019; Nagabandi et al., 2018a). However, using RL to solve practical problems is still a challenging problem due to the low sampling efficiency because, in many scenarios, data availability is limited, or acquiring sufficient data through unrestricted trial-and-error attempts is prohibitively expensive.

Meta-reinforcement learning (meta-RL) can overcome some limitations of existing human-designed RL algorithms (Finn, Abbeel, & Levine, 2017a) by reducing data consumption between different tasks. Meta-RL algorithms exploit shared structure among tasks (such as meta-policy, meta-value function, and meta-dynamics) during meta-training, amortiz-

ing the cost of learning across tasks and enabling rapid adaptation to new tasks during meta-testing from only a small amount of experience. However, it is important to note that meta-learning typically requires a significantly larger amount of data compared to standard learning, as it involves training an entire learning algorithm across multiple tasks, which exacerbates data consumption during online interactions. Consequently, this increased data requirement restricts the applicability of meta-learning in many real-world scenarios.

Developing offline meta-RL (OMRL) methods would enable such methods to leverage existing data from any source, making it easier to scale to real-world problems where large amounts of data might be necessary to generalize broadly. Usually, the offline dataset is collected from multiple tasks using different behavior policies. The training agent aims to learn a meta-policy, which is able to adapt to unseen tasks efficiently. Previous works (Li, Yang, & Luo, 2020; Dorfman, Shenfeld, & Tamar, 2020; Yuan & Lu, 2022) extend context-based meta-RL to the offline setting and propose to use a context encoder to learn task representations from the collected trajectories, then feed the latent codes to policy network to help the decision.

Though considered to be better interpretable and proved to be more effective than the gradient-based method (Mitchell et al., 2021) in OMRL, the context-based (Li et al., 2020) OMRL approach still suffers from two problems: one is the distribution shift problem inherent in the offline setting (Mitchell et al., 2021), and the other is that collected trajectories are affected by both the behavior policy and the task (Yuan & Lu, 2022). Although previous context-based OMRL work (Li et al., 2020; Yuan & Lu, 2022) employed contrastive learning or mutual information techniques to enhance task representations, their learning objectives primarily focused on discriminating trajectories from different tasks rather than explicitly addressing the differences between tasks themselves. As a result, these approaches may not effectively eliminate the influence of behavior policies and may exhibit suboptimal performance on tasks beyond the training range.

To overcome these limitations, inspired by model generalization (Lee et al., 2020) and personalized model (Agarwal et al., 2021) approaches, we propose a new framework called **P**erson**a****L**ized **D**ynamic based on the **M**eta-**M**odel (PLDMM) to deal with OMRL problem. In more detail, we propose an encoder- decoder framework, in which the encoder is expected to infer task-specific parameters through context (system recognition) and generate latent vectors that assist the decoder in making more accurate predictions. The decoder is responsible for learning the shared structure of the transition function and predicting the next state using an input tuple comprising the latent vector, state, and action. We refer to the decoder as the meta-model, and the decoder utilizing the latent vector output by the task inference module as a fixed input is referred to as the personalized model.

For instance, Equation (1) represents the parameterized transition function of the MountainCar environment, where the gravity g_n serves as the parameter. In our framework, the encoder is responsible for inferring the task-specific parameter g_n, while the decoder captures the shared components of the transition function. Our task inference module aims to improve predictions by incorporating task information, thereby reducing the impact of the behavior policy to a certain extent. We propose to use the transformer structure as the encoder to capture task-specific information in the trajectory better. Notably, the training process of the task inference module does not require task labels, which enhances the model's generalization capability beyond the training range. We refer to this training ap-

proach as the unsupervised fashion, in contrast to the supervised fashion that necessitates label supervision at the latent vector level. Once we obtain the task-specific model, it can be combined with offline algorithms to mitigate the distribution shift problem inherent in the offline setting through the model's generalization abilities (Yu et al., 2020).

$$s'_{n1} = s_{n1} + s_{n2} - \frac{g_n \cos(3s_{n1})}{2} + \frac{a_n}{2}, \quad s'_{n2} = s_{n2} - g_n \cos(3s_{n1}) + a_n. \tag{1}$$

We demonstrate the effectiveness of our method on a variety of simulated control tasks from OpenAI gym (Brockman et al., 2016) and MuJoCo physics engine (Todorov et al., 2012). Compared with previous model-based adaptation methods, our method has a lower prediction error, performs better when conducting model predictive control (MPC) (Garcia et al., 1989), and can even be directly used to train the on-policy algorithms. In widely recognized OMRL benchmarks, PLDMM consistently achieves superior performance and demonstrates higher learning efficiency than previous OMRL methods.

Our primary contributions are three-fold:

1) We propose a novel personalized dynamic based on the meta-model (PLDMM) to deal with OMRL problem. To the best of our knowledge, this is the first context-based method to achieve fully offline reinforcement learning adaptation through dynamic models.

2) We propose an unsupervised fashion to train a transformer-based task-inference module, which not only circumvents the challenge of acquiring labels but also exhibits superior generalization beyond the training range compared to existing supervised methods.

3) We experimentally demonstrate on a variety of benchmarks that our PLDMM outperforms prior dynamic model generalization methods and OMRL algorithms. We also concluded that transformer can better extract task-related information in trajectories and using task-specific information as additional input to the model rather than the policy has better interpretability and better results.

2. Related Work

2.1 Offline Reinforcement Learning

Offline reinforcement learning allows policy learning from data collected by arbitrary policies, increasing the sample efficiency of RL. The key issues in offline reinforcement learning are distributional shift and value overestimation. Recent works (Kumar et al., 2020; Yu et al., 2021) propose the use of conservative policy estimation to solve the above issues. And model-based offline methods (Yu et al., 2020, 2021) prove that the generalization of the learned model itself can mitigate the distribution shift problem. Our study follows the offline RL setting but focuses on learning a meta-model that can be quickly transferred to the test task from offline multi-task data.

2.2 Offline Meta-reinforcement Learning

Meta-reinforcement learning algorithms can be classified into gradient-based and context-based based on specific implementation forms. The gradient-based algorithm is based on the policy gradient and learns the process of gradient policy ascent, such as the MAML series of methods (Finn et al., 2017a; Rajeswaran et al., 2019), and the context-based method (Finn et al., 2017b; Kaushik et al., 2020) needs to infer the task through the context. Sodhani et al. (2021) and Humplik et al. (2019) are both context-based methods that use task representation as an additional input to the policy network. They require additional task description information to train the task recognition module separately in a supervised way. However, task descriptions are not always available, and this supervised training method will lead to poor generalization outside the training range. Our method uses the output of the recognition module as an additional input to the dynamic model and trains the recognition module in an unsupervised manner without requiring task information. Lin et al. (2022) explores model-based offline reinforcement learning and regularization policy optimization, learning a meta-model and a meta-policy in a gradient-based way for safely exploring out-of-distribution states, solving the problem of over-dependence of learning strategy on offline data quality. Our method is also inspired by this paper. Learning an accurate context-based model with a few trajectories can also help safe exploration and solve the contradiction between exploration and utilization in offline reinforcement learning.

2.3 Dynamic Model Generalization

Due to the time-consuming and labor-intensive model training, in recent years, the generalization of model-based reinforcement learning has attracted widespread attention. Sanchez-Gonzalez et al. (2018) proposed to use graph networks to model dynamic models to improve the generalization of the model. It has the capability of system recognition. Nagabandi et al. (2018b) studies model-based meta-reinforcement learning, where the function of the meta-learner is to vary the parameters of the model by sensing changes in dynamics. Nagabandi et al. (2018a) proposes that the dynamical model can be adjusted according to the most recent trajectory, by cyclic model hidden variables, or by changing the parameters of the model with few gradient steps. However a problem with these approaches is that the model is responsible for both adjusting and adapting to new next steps. Our approach separates these two functions, using an encoder dedicated to identifying the system, and a general model to make predictions based on this identified latent vector. The closest thing to our general approach is Persim (Agarwal et al., 2021), which builds a personalized model for each task. However, this method requires an additional task index as additional input and there is no explicit connection between the models of each task.

3. Background

We formulate the problem as Hidden-Parameter MDPs (HiP-MDPs) (Doshi-Velez & Konidaris, 2016; Zhang et al., 2020), which can be defined by a tuple $\mathcal{M} : \langle \mathcal{S}, \mathcal{A}, \Theta, T_\theta, R_\theta, \gamma, P_\Theta \rangle$ where $\mathcal{S}$ is a finite state space, $\mathcal{A}$ a finite action space, T_θ describes the transition distribution for a specific task described by task parameter $\Theta \sim P_\Theta$, R_θ is the reward function, γ is the discount factor, and P_Θ the distribution over task parameters. This defines a family of

MDPs, where each MDP ($\mathcal{M}_n$), which is also called a task, is specified by the parameter $\theta_n \sim P_\Theta$. Note that the parameters θ_n may be in the transition function or in the reward function. Given N training tasks $\{\mathcal{M}_i\}_{i=1}^N$ and trajectories from each task D_{train}^i, we hope to learn a task inference module and a meta-model. When encountering an unseen task $\mathcal{M}$ with a task specified parameter θ ($\theta \sim P_\Theta$) and few trajectories D_{test} from it, we can infer the parameter θ and construct a personalized dynamic model based on the meta-model to represent the corresponding MDP.

With the learned task-specific model, we can derive policies using various approaches such as model predictive control (MPC) (Garcia et al., 1989), training online algorithms like PPO (Schulman et al., 2017) and combining with an offline algorithm COMBO. COMBO (Yu et al., 2021) is a well-performing model-based offline reinforcement learning algorithm that combines model-based policy optimization with conservative policy evaluation. Specifically, COMBO first learns a dynamic model through the offline dataset, and then the policy is trained using the offline dataset and model-generated rollouts. We get the policy by replacing the model in the COMBO with our personalized dynamic.

4. PLDMM: Personalized Dynamic based on the Meta-Model

In this work, we hope that based on the meta-model, we can quickly obtain the personalized model of the new and unseen dynamic through a small number of trajectories and then conduct a model-based algorithm with the personalized model to derive RL policy. We present **P**erson**a**L**i**zed **D**ynamic based on the **M**eta-**M**odel (PLDMM), which is a novel framework for rapid adaptation and can get models with better predictions and policies with higher rewards.

As in HiP-MDPS introduced earlier, different tasks can be distinguished by different transition functions or different reward functions. Among them, the problem of changing the transition function is more common in practical scenes, and it is also recognized as relatively difficult to solve. Therefore, in the Methods section, we pay more attention to the problem of changing parameters in the transition functions and state that with only a few minor changes (described in detail in experiments on tasks with altered reward functions), the framework can be translated into a method for solving the changed reward function.

Our framework can be divided into three stages: a) the training phase: training the encoder (system identification module) and the decoder (meta-model module) with D_{train}; b) the adaptation phase: using D_{test} to get a personalized model that can predict the transition function of test dynamic; c)the policy generation phase: using model-based algorithms to get a policy via this personalized model.

4.1 Training Phase

The goal of this phase is to learn an encoder E_d that can learn an implicit representation Z_d of the dynamic-specific parameter and a decoder D_d that learns the general structure of the transition function and can predict the next state $\widehat{s_{t+1}}$ with the fed data tuple of (Z_d, s_t, a_t).

It can be seen from the previous problem setting and application scenarios that we have no way to obtain the parameters of the dynamics directly. Thus, we have to use a module to estimate the dynamic-specific parameter or its implicit representation. As

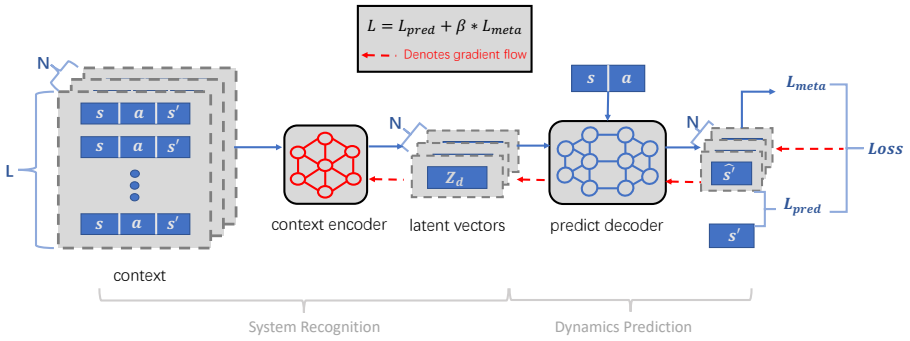


Figure 1: Illustrations of our framework. We divide the entire training process into two parts according to the functions of the modules: the encoder for system recognition and the decoder for prediction. The network weights of both parts are updated by the same *Loss*, which is calculated by L_{meta} and L_{pred}. The blue arrows represent the training process, and the red arrows represent the gradient flow.

in previous reinforcement learning work, we use tuples of (s_t, a_t, s_{t+1}, r_t), which is called context, for system identification. Slightly different in this part, we pay more attention to the environment where the dynamic model changes (the state transition function is different), and the reward function is the same in multiple tasks and independent of task parameters. Thus we use tuples of (s_t, a_t, s_{t+1}) as context. Transformer (Vaswani et al., 2017) is an attention-based model that is often used in sequence modeling. We make changes based on the transformer Pytorch implementation, use the encoder part, set the position encoding to None, and add an FC layer to convert the encoded word vector into a latent vector dimension. By omitting positional encoding, the input transitions lack any temporal ordering, thus preserving the Markov property, emphasizing their dynamic relationships rather than their order or position, and making the model more flexible and better able to generalize to unseen data where the order or position of inputs might differ. Our method selects L joint context points as a context sequence$\{(s_{j+i}, a_{j+i}, s_{j+i+1})\}_{i=0}^{L-1}$ from the j-th step of an offline trajectory, passes it through the context encoder E_d to produce the latent vector Z_d. The position of the context sequence can be fixed or randomly selected. In the specific implementation, we randomly select the context of N positions to make an ensemble like$\{\{(s_{j_n+i}, a_{j_n+i}, s_{j_n+i+1})\}_{i=1}^{L}\}_{n=1}^{N}$. This ensemble performs as the context of the whole trajectory. As described above, a piece of training data can be formatted as:

$$(\{\{(s_{j_n+i}, a_{j_n+i}, s_{j_n+i+1})\}_{i=1}^{L}\}_{n=1}^{N}, s_t, a_t, s_{t+1})$$

The decoder, which is implemented as a feed-forward network, takes the state s_t, action a_t, and dynamics latent vector Z_d as inputs and predicts the next state s_{t+1}. On account of the ensemble of context, we will get an ensemble of latent vector Z_d and then a latent vector of s_{t+1}. Formally,

$$\{Z_{d_n}\}_{n=1}^{N} = E_d(\{\{(s_{j_n+i}, a_{j_n+i}, s_{j_n+i+1})\}_{i=1}^{L}\}_{n=1}^{N}; \theta)$$
$$\{\widehat{s_{t+1}}\}_{n=1}^{N} = D_d(\{Z_{d_n}\}_{n=1}^{N}, s_t, a_t; \phi).$$

As in the previous implementation of the model-based reinforcement learning algorithm, we model the prediction result of a dynamics model $\widehat{M}$ as a Gaussian distribution, output the mean and variance of the distribution, and use the maximum likelihood estimation to calculate the prediction loss $L_{pred} = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\left[\log \widehat{M}\left(s' \mid s, a\right)\right]$. For the convenience of later expression, we use $\widehat{s_{t+1}}$ to represent the mean of each Gaussian distribution. Noting that each one of the prediction results is supposed to have the same value and is equal to the next state s_{t+1}, thus we compute the standard deviation of multiple results as meta-loss $L_{meta} = Std(\{\widehat{s_{t+1}}\}_{n=1}^{N})$. L_{meta} forces the encoder fed with the context on the same trajectory to produce latent vectors that have the same effect on predicting the next state $\widehat{s_{t+1}}$. The final loss is the weighted sum of the two losses, and the coefficient is β, which can be expressed as:

$$Loss = L_{\text{pred}} + \beta * L_{meta} \tag{2}$$

The parameters θ and ϕ of the encoder and decoder are both trained to minimize the final loss $Loss$. We have not used any information on the dynamic-specific parameter or IDs of dynamics but only forced the encoder to produce latent vectors that can help better predict the next state. Thus, we call this training method an unsupervised training fashion, which is the opposite of the method of using the label to make the latent vectors' distance as large as possible. Figure 1 shows an overview of the training phase.

Since the reward function does not vary with the parameters of the transition functions and is unified across all tasks, we separately train a general network for predicting reward values, like $\widehat{r} = f(s_t, a_t; \mu)$. We use the same network structure as the decoder, output the mean and variance of a Gaussian distribution, and update the network parameters using maximum likelihood estimation $\mathbb{E}_{\mathcal{D}}\left[\log f_\mu\left(r \mid s, a\right)\right]$. Due to the large amount of data from multiple tasks used for training, the resulting reward function predictions are relatively accurate.

4.2 Adaptation Phase

In the training phase, we obtain three main components: an encoder responsible for system identification, a decoder that represents the general aspects of the transition function, and a reward prediction model denoted as f_μ. During the adaptation phase, our objective is to derive a model that is suitable for the specific test dynamic environment using only a small number of trajectories.

As mentioned earlier, we can extract a context sequence $\{(s_{j+i}, a_{j+i}, s_{j+i+1})\}_{i=0}^{L-1}$ from the trajectories in the test dataset D_{test}. By utilizing the trained encoder E_d, we can obtain the latent vector Z_{test} corresponding to the test dynamic. Next, we fix Z_{test} as an input to the decoder component D_d, which represents the general part of the transition function. This process results in a personalized dynamic based on the meta-model, referred to as PLDMM. Specifically, our PLDMM takes the current state s_t and action a_t as inputs and predicts the subsequent state $\widehat{s_{t+1}}$. With the learned reward function and tuples of $(s_t, a_t, \widehat{s_{t+1}})$, we are able to get the MDPs like $(s_t, a_t, \widehat{s_{t+1}}, \widehat{r_t})$, which can be used to train RL policy.

In fact, we encapsulate the model into a gym format so that the environment can use the reset and step functions. When the reset function is executed, the current state of

the virtual environment will be set as one of the first states from the test trajectories. In addition, since our D_{test} may include more than one trajectory, we can also get multiple context sequences and then naturally get the ensemble of multiple models. For example, if we have K trajectories in D_{test}, then with the ensemble of K PLDMM, K prediction results can be obtained at one time:

$$\begin{aligned}\{Z_{test_n}\}_{n=1}^{K} &= E_d(\{\{(s_{j_n+i}, a_{j_n+i}, s_{j_n+i+1})\}_{i=1}^{L}\}_{n=1}^{K}) \\ \{\widehat{s_{t+1}}\}_{n=1}^{K} &= D_d(\{Z_{test_n}(fixed)\}_{n=1}^{K}, s_t, a_t) \\ \widehat{r_t} &= f(s_t, a_t; \mu).\end{aligned}$$

4.3 Policy Generation Phase

Since Z_{test} is fixed as input, only the state and action at the current time are needed to predict the next state. This characteristic allows us to utilize our PLDMM model in various ways, particularly for policy generation.

Given the accuracy and encapsulation of our PLDMM as virtual environments, we can employ the model in three different manners: model predictive control (MPC), training online reinforcement learning (RL) algorithms like Proximal Policy Optimization (PPO), and combining it with offline algorithms.

We utilize the MPC method (Agarwal et al., 2021) to generate policy. Specifically, we sample C candidate action sequences of length h, which we denote as $\left\{a_1^{(i)}, \ldots, a_h^{(i)}\right\}_{i=1}^{C}$. The actions are sampled using cross entropy for environments with continuous action spaces or random shooting for environments with discrete action spaces. Using an ensemble of K simulators, we evaluate the average reward $r^{(i)}$ obtained by executing the i-th action sequence across the K simulators. To be precise,

$$\widehat{r^{(i)}} = \frac{1}{K} \sum_{k=1}^{K} \sum_{t=1}^{h} f\left(\widehat{s}_t^{(k,i)}, a_t^{(i)}\right)$$

where $\widehat{s}_t^{(k,i)}$ is the predicted trajectory according to the k-th simulator and the sequence of actions $\left\{a_1^{(i)}, \ldots, a_h^{(i)}\right\}$, and f is the reward function. Finally, we choose the first element from the sequence of actions with the best average reward, i.e., the sequence $\left\{a_1^{(i^*)}, \ldots, a_h^{(i^*)}\right\}$, where $i^* = \text{argmax}_{i \in [C]} \widehat{r^{(i)}}$.

To train online RL algorithms like PPO, we encapsulate the model into a gym format so that the environment can use the reset and step functions. When the reset function is executed, the current state of the virtual environment will be set as one of the first states from the test trajectories. Then, PPO policies can be trained by constantly interacting with the simulators.

When combined with the offline algorithm COMBO, we replace the prediction model $\widehat{T}_\theta(s' \mid s, a)$ obtained through offline data supervision training with the PLDMM.

5. Experiments

In this section, we evaluate the performance of our PLDMM method to answer the following questions:

- Does our context encoder trained in an unsupervised manner extract meaningful contextual information?
- Does PLDMM make better predictions and have better predictive performance in environments outside the training range than previous work on model generalization?
- Do policies obtained via PLDMM outperform other model generalization methods?
- How does the combination of PLDMM and offline model-based algorithm compare with other OMRL algorithms?

Besides, we verify the effectiveness of the components in our framework through ablation experiments.

5.1 Setup

We evaluate PLDMM on various benchmark environments from OpenAI Gym (Brockman et al., 2016). A detailed description of each environment can be found in Appendix B. We used the same experimental environment as the compared baselines for a fair comparison.

5.1.1 Comparison with Model Generalization Methods

Continuing the settings in the CaDM (Lee et al., 2020) and Persim (Agarwal et al., 2021), We modify the dynamics parameters in the gym environment and assume that the reward functions and done functions are known. For each environment, we choose one of the dynamic parameters as variable parameters, such as gravity in MountainCar, force magnitude and pole length in CartPole, mass-scale, and damping-scale in HalfCheetah, mass-scale, and damping-scale in SlimHumanoid. We define a training range for each parameter and select 5 tasks within that range as the training tasks. During testing, we include the original task and four unseen tasks. Two unseen tasks are within the training range, while the other two are outside the training range. These tasks are used to evaluate the model's prediction error and the average reward obtained by MPC. Additionally, we employ these tasks as evaluation tasks for offline algorithms.

As for the offline dataset, in order to explore the performance of our method under different data qualities, we use a probabilistic selection parameter ϵ=0,0.2,0.4,1.0 respectively to get four types of offline datasets. See Appendix B for details about collecting the dataset and the average reward achieved across all training tasks using this procedure.

5.1.2 Comparison with Offline Meta-RL Methods

Our experiments considered two categories of environments commonly used in OMRL methods. The first category includes Walker-Param and Hopper-Param, which are multi-task MuJoCo benchmarks where tasks differ in *transition dynamics*. Each task has randomized physical parameters such as body mass, inertia, damping, and friction. The agent needs to

adapt its behavior to accommodate the varying dynamics and successfully accomplish the task.

The second category comprises Point-Robot, Half-Cheetah-Vel, and Ant-Dir, where tasks differ in *reward functions* while the transition dynamics remain the same across tasks. In these environments, the variable parameters in the reward function are determined by the target speed or position. Thus, our objective is to identify these target speed or position parameters. During the training phase, the encoder is trained to identify a parameter that aids the decoder in predicting reward values more accurately. Simultaneously, the decoder learns a general dynamic function to predict the state of the next step. The formal setup for these environments is as follows:

$$\{Z_{d_n}\}_{n=1}^{N} = E_d(\{\{(s_{j_n+i}, a_{j_n+i}, s_{j_n+i+1}, r_{j_n+i})\}_{i=1}^{L}\}_{n=1}^{N}; \theta)$$
$$\{\widehat{r}_t\}_{n=1}^{N} = D_d(\{Z_{d_n}\}_{n=1}^{N}, s_t, a_t; \phi)$$
$$\widehat{s_{t+1}} = f(s_t, a_t; \mu).$$

For each environment, 20 training and 20 testing tasks are sampled from the task distribution. We use PPO (Schulman et al., 2017) to train a single-task policy independently on each task. The replay buffers are collected as offline datasets.

5.2 Baselines

We compare with five state-of-the-art RL algorithms; (i) two from the model generalization literature and their variants; (ii) three from the offline meta-RL literature. The baseline details are as follows:

- *Vanilla CaDM + PE-TS CaDM* (Lee et al., 2020). CaDM tackles heterogeneity by learning a context vector using the recent trajectory of a given agent, with a common context encoder across all agents. Vanilla CaDM combined CaDM with Vanilla DM, which models the dynamics model as Gaussian, in which the mean is parameterized by MLPs with 4 hidden layers of 200 units each and Swish activations, and the variance is fixed. PE-TS CaDM combined CaDM with PE-TS, which used 5 bootstrap models for the ensemble and 20 particles for trajectory sampling.

- *Persim + Persim-M* (Agarwal et al., 2021). Persim proposes to build a personalized model for each environment and use model predictive control on the built model. However, for Persim, there are multiple environments for training, but there is only one trajectory in each environment, and the one-hot encoding of each environment needs to be obtained. We consider two Persim baselines: (i) Persim-Original (Persim), in which we select 100 tasks at equal intervals within the training range and collect offline data of a trajectory in each task. The task's id is set to the sequence number of the task in the training range. (ii) Persim-Modified (Persim-M), in which we modify Persim to our setting that only has 5 training tasks with 100 trajectories each.

- *MACAW* (Mitchell et al., 2021). MACAW is a gradient-based model-free OMRL algorithm that combines gradient-based meta-learning and off-policy value-based RL and possesses three key properties: sample efficiency, offline meta-training, and consistency at meta-test time.

- *MerPO* (Lin et al., 2022). MerPO is a gradient-based model-based OMRL algorithm that learns a meta-model for efficient task structure inference and an informative meta-policy for the safe exploration of out-of-distribution state actions.
- *FOCAL* (Li et al., 2020). FOCAL is a context-based model-free OMRL algorithm that proposes a novel negative-power distance metric on bounded context embedding space for efficient task inference. Then use the identified latent vector as an additional parameter of the policy network in the meta-test phase.

5.3 Model Learning: Prediction Error

In order to evaluate the effectiveness of our method and compare it with the baseline approaches, we focus on the rapid adaptation of dynamic models based on the meta-model. The same evaluation criterion is applied to the baseline methods as well. Specifically, we assess the prediction accuracy of the generated models in the test environment.

For each test task, we first generate the respective models using the algorithm process of each method. Then, we simultaneously conduct rollout simulations in the generated models and the test environment, using random actions. At each step, we calculate the mean-squared error (MSE) between the predicted state and the actual next state and accumulate these errors over the course of an episode (until the maximum number of steps or a termination condition is reached). We repeat this procedure 200 times for each test environment and compute the mean and standard deviation of the mean MSE.

Data	Method	Dynamic1(0.3)	Dynamic2(0.6)	Dynamic3(1.0)	Dynamic4(1.4)	Dynamic5(1.7)
Pure	Persim	4.02(±2.305)	3.698(±2.311)	3.115(±2.794)	3.484(±3.032)	3.635(±2.993)
	Persim-M	64.975(±60.354)	11.002(±8.315)	3.576(±2.074)	4.894(±4.879)	58.088(±30.926)
	Vanilla CaDM	6.247(±1.596)	3.878(±0.895)	1.795(±0.646)	2.904(±0.815)	3.88(±1.013)
	PE-TS CaDM	5.321(±1.74)	3.14(±0.911)	1.827(±0.849)	1.191(±0.537)	**3.109(±0.856)**
	OURS	**3.883(±1.079)**	**2.499(±1.163)**	**1.721(±0.905)**	**1.182(±0.288)**	3.781(±0.682)
Pure-ϵ-20	Persim	1.155(±0.167)	1.181(±0.133)	1.157(±0.161)	1.123(±0.189)	1.108(±0.205)
	Persim-M	5.247(±4.282)	2.989(±0.163)	5.827(±5.263)	13.694(±6.255)	46.063(±29.753)
	Vanilla CaDM	6.017(±1.022)	3.587(±1.13)	2.058(±0.922)	2.259(±0.837)	3.438(±1.23)
	PE-TS CaDM	4.825(±1.959)	2.918(±1.644)	1.976(±1.108)	2.059(±0.736)	2.967(±1.0)
	OURS	**1.122(±0.043)**	**1.121(±0.756)**	**1.169(±0.768)**	**0.981(±0.059)**	**1.015(±0.038)**
Pure-ϵ-40	Persim	1.073(±0.145)	1.131(±0.169)	1.081(±0.164)	1.003(±0.153)	0.969(±0.163)
	Persim-M	2.726(±0.133)	2.467(±0.689)	2.013(±0.128)	2.089(±0.108)	2.176(±0.149)
	Vanilla CaDM	5.965(±2.23)	3.613(±1.21)	2.147(±1.433)	2.552(±0.88)	3.69(±1.163)
	PE-TS CaDM	5.046(±1.53)	3.027(±1.399)	1.8(±0.767)	1.926(±1.125)	3.043(±0.957)
	OURS	**1.008(±0.913)**	**0.754(±0.545)**	**0.672(±0.044)**	**0.744(±0.035)**	**0.793(±0.063)**
Random	Persim	1.024(±0.124)	1.053(±0.151)	0.99(±0.177)	0.93(±0.206)	0.889(±0.207)
	Persim-M	3.277(±0.145)	2.736(±0.13)	2.74(±0.119)	3.121(±0.136)	4.184(±0.35)
	Vanilla CaDM	4.465(±1.171)	4.127(±1.387)	3.833(±1.308)	4.001(±1.087)	4.096(±1.523)
	PE-TS CaDM	4.176(±0.737)	2.727(±0.362)	1.967(±0.234)	2.016(±0.662)	2.761(±0.604)
	OURS	**0.847(±0.317)**	**0.537(±0.017)**	**0.591(±0.027)**	**0.644(±0.044)**	**0.71(±0.036)**

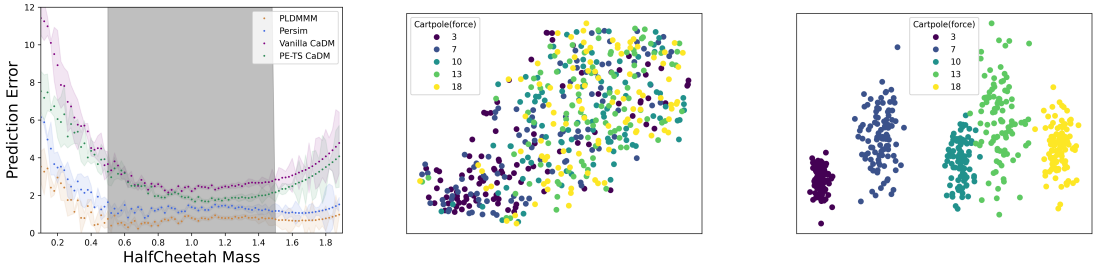
Table 1: Prediction Error: HalfCheetah(Mass Scale)

Due to space limitations, we only show the results of one environment in the text. For more results, see Appendix D. The results of predictions are summarized in Table 1 for HalfCheetah (Mass). PLDMM consistently outperforms other algorithms for most test experiments across tasks and data quality. It can be seen from Table 1 that whether PLDMM is in the training range (Dynamic 2,3,4) or outside the training range (Dynamic 1,5), the prediction error is smaller, which proves the prediction ability and generalization ability of PLDMM. Furthermore, when examining the prediction error variance longitudinally with

changes in data quality, PLDMM demonstrates significantly smaller variances than other methods. This suggests that PLDMM can better capture differences in dynamic parameters and mitigate the impact of behavior policy. This also highlights the effectiveness of the task inference module in PLDMM.

We conducted tests across all tasks to provide a more intuitive visualization of PLDMM's generalization performance and plotted the results in Figure 2a. From the figure, it is evident that PLDMM exhibits superior prediction performance within the training range and demonstrates a slower increase in prediction error for out-of-distribution (OOD) tasks outside the training range. This can be attributed to the unsupervised training of our task inference module, which helps the decoder make more accurate predictions by identifying task-relevant information from the trajectory. In contrast, methods that rely on task IDs may perform comparably to PLDMM within the training range but exhibit significantly poorer performance outside the range.

Another feature of our method is to use of an unsupervised way to train the system recognition module (encoder). In order to prove that our method can capture dynamic information, we perform dimensionality reduction processing on the hidden vector obtained by the encoder before and after training through the same context and draw t-SNE diagrams. We can clearly observe that compared with an untrained encoder in Figure 2b, the latent vector after training shows obvious regularity in Figure 2c. The points are more organized and form distinct clusters. This indicates that the trained encoder has learned to extract relevant features and encode them into a latent space that captures the underlying structure of the dynamics. These t-SNE plots provide visual evidence that our method, through unsupervised training of the system recognition module, is able to effectively capture dynamic information and learn meaningful representations that can aid in system identification and adaptation.



(a) generalization (b) Before training (c) After training

Figure 2: (a)Prediction error in Halfcheetah (Mass Scale).Gray boxes indicate the training range. (b)t-SNE visualization of latent context vectors inferred by encoder before training extracted from contexts collected in various tasks in CartPole (force). Embedded points from tasks with the same parameter have the same color. (c)t-SNE visualization of latent context vectors inferred by encoder after training.

5.4 Policy Performance: Average Reward

In this section, we evaluate the average reward achieved by PLDMM compared to several state-of-the-art model generation RL methods. Since previous methods, such as Persim and CaDM both use MPC to obtain policies, after obtaining PLDMM, we continue to use MPC in their own way. We include an additional baseline, "PLDMM + PPO", where we use PLDMM as a simulator, using the initial state of the test trajectories as the start, combined with the known reward function and done function, to train the PPO algorithm online.

For model-based methods, we evaluate the performance of each method using the average reward over 20 episodes. We repeat each experiment five times and report the mean and standard deviation. For "PLDMM + PPO", we train for 400 epochs each time and repeat five times like the model-based method. The results of the average reward in SlimHumanoid (Mass Scale)can be found in Table 2.

Analyzing the results in Table 2, we observe that PLDMM performs the best in almost all cases of data quality when using MPC to obtain the policy. However, it is worth noting that all methods perform relatively poorly in the Pure case. We attribute this to the limited state coverage in the dataset collected by the well-performing policy, making it challenging for the model-based methods to learn an effective state transition function. Moreover, SlimHumanoid presents difficulties for model-based methods due to its high-dimensional state and action spaces (45-dimensional observations and 17-dimensional actions). Interestingly, "PLDMM + PPO" achieves better or comparable performance compared to other methods using MPC, indicating that the PLDMM obtained can serve as an effective simulator for training online algorithms.

Data	Method	Dynamic1(0.6)	Dynamic2(0.9)	Dynamic3(1.0)	Dynamic4(1.1)	Dynamic5(1.4)
Pure	Persim	182.9(±55.7)	306.4(±19.3)	565.8(±16.6)	431.8(±59.8)	138.8(±71.3)
	Persim-M	51.1(±87.3)	175.9(±18.7)	488.5(±15.3)	341.6(±11.7)	64.6(±77.9)
	Vanilla CaDM	152.6(±26.1)	251.4(±44.3)	455.8(±25.2)	252.3(±68.4)	122.4(±55.6)
	PE-TS CaDM	203.0(±32.2)	347.9(±25.8)	623.4(±22.4)	294.8(±32.9)	169.5(±69.8)
	OURS	**272.5(±31.3)**	**355.3(±10.7)**	**665.5(±32.4)**	**442.7(±119.0)**	**274.3(±81.9)**
	OURS + PPO	266.9(±31.9)	327.6(±13.6)	530.7(±41.8)	405.6(±76.1)	267.1(±84.4)
Pure-ϵ-20	Persim	270.6(±54.3)	1549.0(±38.9)	2023.5(±25.2)	1245.1(±131.8)	805.2(±29.8)
	Persim-M	-509.8(±72.1)	465.8(±25.2)	1943.5(±15.2)	1309.3(±22.9)	795.2(±12.5)
	Vanilla CaDM	-582.4(±39.8)	252.2(±27.0)	1567.3(±15.5)	349.3(±13.9)	51.4(±37.5)
	PE-TS CaDM	478.2(±91.8)	834.5(±25.0)	2099.0(±24.6)	1412.0(±44.2)	307.2(±49.4)
	OURS	**575.4(±76.0)**	**1717.1(±62.9)**	**2975.6(±71.8)**	**1433.3(±106.1)**	**1003.7(±87.5)**
	OURS + PPO	572.8(±102.7)	1670.4(±75.6)	2174.3(±49.6)	1296.2(±75.0)	974.7(±63.7)
Pure-ϵ-40	Persim	1069.9(±32.1)	2357.0(±22.3)	3086.0(±13.9)	2182.8(±18.9)	1264.4(±50.9)
	Persim-M	305.8(±45.8)	2018.3(±28.4)	2110.9(±34.5)	2364.0(±28.1)	178.9(±83.9)
	Vanilla CaDM	-501.6(±78.8)	2215.9(±18.2)	1915.2(±26.0)	1885.3(±44.3)	-292.7(±75.1)
	PE-TS CaDM	930.2(±41.1)	1845.1(±29.2)	2826.7(±17.0)	2370.5(±22.9)	1021.4(±68.1)
	OURS	**1175.4(±168.4)**	**2707.0(±30.6)**	**3830.6(±19.2)**	**2485.5(±33.1)**	**1459.8(±42.6)**
	OURS + PPO	1137.5(±156.3)	2318.8(±27.0)	3327.5(±19.7)	2309.0(±39.1)	1327.4(±59.2)
Random	Persim	473.1(±23.8)	1782.2(±36.7)	1893.3(±15.2)	1987.3(±32.0)	574.9(±65.1)
	Persim-M	-614.2(±42.5)	1208.4(±16.7)	1124.9(±8.1)	1220.4(±15.6)	343.7(±37.8)
	Vanilla CaDM	-1322.9(±64.1)	470.4(±21.5)	767.9(±7.0)	736.1(±15.5)	-485.8(±47.7)
	PE-TS CaDM	161.7(±39.0)	1354.5(±16.3)	1402.9(±15.4)	1839.0(±4.5)	598.0(±29.4)
	OURS	**505.8(±24.0)**	**1960.7(±66.1)**	**2236.4(±14.8)**	**1987.9(±18.3)**	**1261.9(±25.8)**
	OURS + PPO	298.7(±22.9)	2004.6(±101.6)	2030.9(±12.2)	1894.9(±13.3)	1174.2(±31.3)

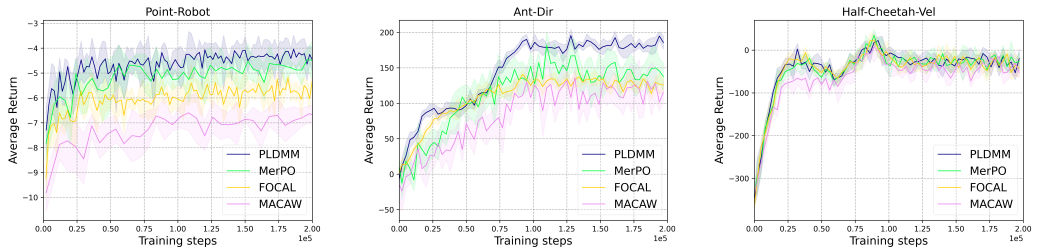
Table 2: Average Reward: SlimHumanoid (Mass Scale)

As a high-level summary, PLDMM outperforms previous algorithms in almost all environments and data quality thanks to the smaller prediction error. These results corroborate

the appropriateness of PLDMM and our overall methodological framework as a principled solution to this challenging yet meaningful setting. Furthermore, experimental results show that our PLDMM can be used in more ways, such as directly training online algorithms as a simulator.

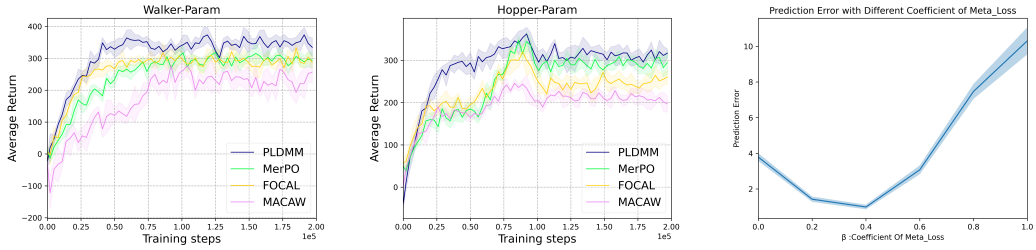
5.5 Comparison with Offline Meta-RL Methods

We followed the experimental environment of the previous OMRL methods. For each environment, 20 training tasks and 20 testing tasks are sampled from the task distribution. Because we save the reply buffer as an offline data set during the training of PPO, we can consider the offline data set to be a mixture of data collected by multiple behavior policies. To evaluate the performance on test tasks, we select five trajectories from the reply buffer of each test task for transfer. Then, the average return over all the test tasks is compared to measure the performance.



(a) (b) (c)

Figure 3: Test returns of PLDMM against the baselines in the environments with different reward functions: Point-Robot(a), Ant-Dir(b), and Half-Cheetah-Vel(c). The shaded region shows a standard deviation across 5 seeds.



(a) (b) (c)

Figure 4: Test returns of PLDMM against the baselines in the environments with different transition dynamics: Walker-Param(a) and Hopper-Param(b). The shaded region shows a standard deviation across 5 seeds. (c)Prediction error with different coefficient of L_{meta}, the shaded region shows standard deviation across 5 experiments

Figure 3 shows the test results in the environments with different reward functions. In Point-Robot, PLDMM and MerPO outperform other baselines. Although the return value changes during the training process are relatively small in this environment, we can still see

that PLDMM's zero-shot performance is the best. PLDMM shows better learning efficiency when its performance is similar to that of MerPO. In Ant-Dir, PLDMM outperforms all the baselines both in average return and learning efficiency. And another context-based method FOCAL shows better learning efficiency than the gradient-based methods. However, in Half-Cheetah-Vel, we cannot distinguish the performance of PLDMM and MerPO; they both outperform FOCAL and MACAW, which shows the advantages of the model-based approach.

Figure 4(a)(b) demonstrates the results in the environments with varying transition dynamics. Compared to the environment where the reward function is changed, PLDMM significantly outperforms baselines in terms of learning efficiency and final performance in these environments. We also found that when the transition function changes, the context-based methods are more efficient than the gradient-based methods, and the model-based methods can finally achieve better performance than the model-free algorithms. This finding can also prove the necessity of our context-model-based approach to solving OMRL problems.

Overall, the experimental results presented in Figures 3 and 4 demonstrate the superiority of PLDMM across different environments with varying reward functions and transition dynamics. PLDMM consistently achieves better learning efficiency and performance compared to the baseline methods, confirming the effectiveness of our context-model-based approach in solving OMRL tasks.

5.6 Ablation Study

5.6.1 Coefficient β

The impact of the weight coefficient β on the prediction error in the final test environment was investigated to verify the necessity of L_{meta} in PLDMM. In the Halfcheetah (Mass-Scale) environment, the coefficients were varied in the range of [0, 1.0]. The model was trained five times, and the average prediction error across all five test tasks was recorded.

Figure 4c presents the results of the ablation experiment. It is evident that selecting an appropriate coefficient leads to a significant decrease in the prediction error. However, as the coefficient increases, L_{meta} starts to impact the original prediction objective, resulting in an increase in the prediction error.

This experiment provides evidence for the effectiveness of our proposed L_{meta} loss term. It also guides us in choosing appropriate coefficients for all experiments. In this case, the optimal coefficient was determined to be $\beta = 0.4$.

The results of the ablation experiment further highlight the importance of incorporating L_{meta} in PLDMM, as it helps enforce consistency among latent vectors generated by the encoder when provided context from the same trajectory.

5.6.2 Transformer-Based Encoder

In order to prove that the transformer as a task recognition module can better obtain task-specific information from offline trajectories, we replaced the transformer in the encoder with MLP and compared the corresponding prediction errors and MPC results. Specifically, for PLDMM-MLP, the encoder is an MLP of 2 hidden layers with 200 hidden-width for each layer, and the remaining parts are exactly the same as PLDMM. We also change the

length of the context (5, 10, 20) to observe the impact of the context length on the results. We conduct experiments on all data qualities in all environments and report the average prediction error and MPC performance across all environments' data qualities.

Method	MountainCar		CartPole		HalfCheetah		SlimHumanoid	
	Prediction Error	Average Reward	Prediction Error	Average Reward	Prediction Error	Average Reward	Prediction Error	Average Reward
PLDMM(5)	0.68	-100.9	0.83	180.6	1.87	949.5	4.75	1032.5
PLDMM(10)	0.65	-84.2	0.76	189.3	1.24	1061.7	3.04	1696.3
PLDMM(20)	0.60	-79.6	0.70	192.3	0.79	1688.8	2.05	1883.2
PLDMM-MLP(5)	1.33	-134.2	1.55	131.8	6.04	263.5	12.74	346.9
PLDMM-MLP(10)	1.14	-110.4	1.46	140.7	6.71	315.0	13.21	461.0
PLDMM-MLP(20)	1.37	-139.2	1.37	133.7	5.84	436.8	18.82	354.5

Table 3: The impact of encoders based on transformer and MLP and different context lengths on prediction results and MPC performance.

As seen from Table 3, at the same context length L, the recognition module using the transformer structure has significantly lower prediction errors and, therefore, has better MPC results. This also verifies the effectiveness of the encoder with the transformer structure. In addition, as the context length increases, PLDMM recognition error gradually decreases and performs better. In contrast, PLDMM-MLP performs worse, indicating that the transformer is more suitable for identifying task information from long sequence trajectories. We attribute this to the fact that the attention mechanism of the transformer can better extract task-related information in the context. Although the computational complexity of the encoder based on the transformer is $O(L^2)$, while that of the MLP is only $O(L)$. Since the context length we take is small, the actual training time is not much different, but the encoder effect of using the transformer structure is significantly better.

5.6.3 Effect of Unsupervised Task Inference Fashion

The t-SNE diagram shows that the task-specific vectors obtained by the encoder can distinguish different tasks. In this part, we want to explore the representation impacts of the task-specific vectors obtained in an unsupervised fashion. For comparison, we use the other two methods to obtain task-specific vectors:

- *One-hot encoding of task index (One-hot).* We convert the task index into a one-hot vector with the same dimension as Z_d, and fix it as the input of the decoder to obtain the personalized model.
- *Supervised fashion (Supervised).* Like FOCAL (Li et al., 2020), we use the task index as the label and train the encoder separately in a supervised fashion to obtain the task identification module. The contrastive loss to train encoder E_d is written as:

$$\mathcal{L}\left(Z_i, Z_j; E_d\right) = \mathbf{1}\left\{y_i = y_j\right\} \left\|Z_i - Z_j\right\|_2^2 + \mathbf{1}\left\{y_i \neq y_j\right\} \frac{1}{\left\|Z_i - Z_j\right\|_2^2 + \epsilon} \tag{3}$$

  where Z_i and Z_j are latent vectors obtained by encoder E_d, y_i and y_j are task index, and $\epsilon > 0$ is a small hyperparameter added to avoid division by zero. The encoder E_D is then freezed when training decoder D_d.

We test the prediction error and average reward of all tasks under four data qualities in Halfcheetah (Mass Scale) environment and report the average performance within the training range and outside the training range, respectively.

Method	Within training range		Outside training range	
	Prediction Error	Average Reward	Prediction Error	Average Reward
PLDMM	1.02	1235.6	2.42	894.8
One-hot	8.53	197.5	12.83	69.3
Supervised	0.93	1304.3	3.59	625.7

Table 4: The impact of different task identification methods on prediction error and MPC performance.

As seen from Table 4, although one-hot encoding can refer to each task, its performance for unseen tasks is inferior and cannot be used. Compared with the encoder trained by supervised learning, although PLDMM performs slightly worse within the training range, the prediction error and MPC reward value outside the training range are significantly higher, showing more vital generalization ability outside the training range. For model-based methods, the role of task information should be to help the model predict more accurately, rather than distinguishing more accurately at the latent vector level. Excessive supervision makes the task recognition module more accurate in identifying tasks it has seen, but its performance on unseen tasks will be worse.

At the same time, compared to supervised learning methods that require computational complexity proportional to the square of the batch size, the training time of our unsupervised training method is significantly shorter. Therefore, considering the generalization ability and computational consumption, the unsupervised training model we proposed is more suitable for actual generalization scenarios.

6. Conclusion

In this work, we have tackled the challenging problem of OMRL, explicitly addressing the issue of changing transition functions. We formulated the problem as HiP-MDPs and introduced a novel framework called PLDMM, a context-model-based method designed to enable efficient adaptation. Our approach involves training the system recognition module in an unsupervised manner and leveraging the prediction module as a meta-model to generate models in the test environment, followed by policy generation using various methods. Through extensive experiments, we have demonstrated the effectiveness of our approach in terms of generalization, availability, and superiority compared to previous OMRL algorithms.

The utilization of offline data and the improvement of RL algorithm generalizability are crucial for successfully applying reinforcement learning in real-world scenarios. Our research contributes valuable insights to these areas. However, several aspects require further investigation in future work. One such aspect is the development of a more robust world model representation. Additionally, addressing the challenges posed by the changing environments of both transition functions and reward functions would be an important direction for future research. We leave these avenues as open directions for future exploration.

References

Agarwal, A., Alomar, A., Alumootil, V., Shah, D., Shen, D., Xu, Z., & Yang, C. (2021). Persim: Data-efficient offline reinforcement learning with heterogeneous agents via personalized simulators. *Advances in Neural Information Processing Systems*, *34*, 18564–18576.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Dorfman, R., Shenfeld, I., & Tamar, A. (2020). Offline meta learning of exploration. *arXiv preprint arXiv:2008.02598*.

Doshi-Velez, F., & Konidaris, G. (2016). Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations. In *IJCAI: proceedings of the conference*, Vol. 2016, p. 1432. NIH Public Access.

Finn, C., Abbeel, P., & Levine, S. (2017a). Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR.

Finn, C., Yu, T., Zhang, T., Abbeel, P., & Levine, S. (2017b). One-shot visual imitation learning via meta-learning. In *Conference on robot learning*, pp. 357–368. PMLR.

Garcia, C. E., Prett, D. M., & Morari, M. (1989). Model predictive control: Theory and practice—a survey. *Automatica*, *25*(3), 335–348.

Humplik, J., Galashov, A., Hasenclever, L., Ortega, P. A., Teh, Y. W., & Heess, N. (2019). Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. (2019). Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*.

Kaushik, R., Anne, T., & Mouret, J.-B. (2020). Fast online adaptation in robotics through meta-learning embeddings of simulated priors. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5269–5276. IEEE.

Kumar, A., Zhou, A., Tucker, G., & Levine, S. (2020). Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, *33*, 1179–1191.

Lee, K., Seo, Y., Lee, S., Lee, H., & Shin, J. (2020). Context-aware dynamics model for generalization in model-based reinforcement learning. In *International Conference on Machine Learning*, pp. 5757–5766. PMLR.

Li, L., Yang, R., & Luo, D. (2020). Focal: Efficient fully-offline meta-reinforcement learning via distance metric learning and behavior regularization. *arXiv preprint arXiv:2010.01112*.

Lin, S., Wan, J., Xu, T., Liang, Y., & Zhang, J. (2022). Model-based offline meta-reinforcement learning with regularization. *arXiv preprint arXiv:2202.02929*.

Mitchell, E., Rafailov, R., Peng, X. B., Levine, S., & Finn, C. (2021). Offline meta-reinforcement learning with advantage weighting. In *International Conference on Machine Learning*, pp. 7780–7791. PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529–533.

Nagabandi, A., Clavera, I., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., & Finn, C. (2018a). Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*.

Nagabandi, A., Finn, C., & Levine, S. (2018b). Deep online learning via meta-learning: Continual adaptation for model-based rl. *arXiv preprint arXiv:1812.07671*.

Rajeswaran, A., Finn, C., Kakade, S. M., & Levine, S. (2019). Meta-learning with implicit gradients. *Advances in neural information processing systems*, *32*.

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., & Battaglia, P. (2018). Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pp. 4470–4479. PMLR.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, *588*(7839), 604–609.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sodhani, S., Zhang, A., & Pineau, J. (2021). Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning*, pp. 9767–9779. PMLR.

Todorov, E., Erez, T., & Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Yu, T., Kumar, A., Rafailov, R., Rajeswaran, A., Levine, S., & Finn, C. (2021). Combo: Conservative offline model-based policy optimization. *Advances in neural information processing systems*, *34*, 28954–28967.

Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J. Y., Levine, S., Finn, C., & Ma, T. (2020). Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, *33*, 14129–14142.

Yuan, H., & Lu, Z. (2022). Robust task representations for offline meta-reinforcement learning via contrastive learning. In *International Conference on Machine Learning*, pp. 25747–25759. PMLR.

Zhang, A., Sodhani, S., Khetarpal, K., & Pineau, J. (2020). Learning robust state abstractions for hidden-parameter block mdps. *arXiv preprint arXiv:2007.07206*.

Zhang, M., Vikram, S., Smith, L., Abbeel, P., Johnson, M., & Levine, S. (2019). Solar: Deep structured representations for model-based reinforcement learning. In *International conference on machine learning*, pp. 7444–7453. PMLR.

Appendix A. PSEUDO-CODE

Algorithm 1 Training Phase(transition function)

Input: Offline Dataset D_{train}; encoder E_θ; decoder D_θ; return functionf_μ; transition buffer $B = \emptyset$; reward buffer $R = \emptyset$; length and number of context sequence L, N;

Get training data:
for each trajectory T in D_{train} **do**
  select N positions, $\{(j_n)\}_{n=1}^{N}$
  for n=1, N **do**
    sample a context of length L, $\{(s_{j_n+i}, a_{j_n+i}, s_{j_n+i+1})\}_{i=1}^{L}$
  end for
  merge of N context sequences $\{\{(s_{j_n+i}, a_{j_n+i}, s_{j_n+i+1})\}_{i=1}^{L}\}_{n=1}^{N}$
  for each position t in T **do**
    sample a tuple of (s_t, a_t, s_{t+1}, r_t)
    combine into a tuple $T_t = (\{\{(s_{j_n+i}, a_{j_n+i}, s_{j_n+i+1})\}_{i=1}^{L}\}_{n=1}^{N}, s_t, a_t, s_{t+1})$
    get a tuple of reward $R_t = (s_t, a_t, r_t)$
    add data to buffer $B = B \bigcup T_t$; $R = R \bigcup R_t$
  end for
end for
Training:
while not done **do**
  sample a batch of tuples $(T_t)_{t:1\ldots b} \sim B$.
  sample a batch of tuples $(R_t = (s_t, a_t, r_t))_{t:1\ldots b} \sim R$.
  compute latent vectors $\{Z\}_{n=1}^{N} = E_\theta(\{\{(s_{j_n+i}, a_{j_n+i}, s_{j_n+i+1})\}_{i=1}^{L}\}_{n=1}^{N})$
  predict $\{\widehat{s_{t+1}}\}_{n=1}^{N} = D_\theta(\{Z\}_{n=1}^{N}, s_t, a_t)$; $\{\widehat{r_t}\} = f_\mu(s_t, a_t)$
  compute prediction $Loss$ with Eq.(2)
  compute reward loss with $Loss_R = \mathbb{E}_{\mathcal{R}}\left[\log f_\mu\left(\widehat{r_t} \mid s, a\right)\right]$
  update θ and μ with $Loss$ and $Loss_R$
end while

Algorithm 2 Adaptation Phase(transition function)

Input: Offline Dataset in unseen task D_{test}; encoder E_θ; decoder D_θ; return functionf_μ; length of context sequence L; number of ensemble K(the number of trajectory in D_{test});

for each trajectory T in D_{test} **do**
  sample a context of length L, $\{(s_{j_n+i}, a_{j_n+i}, s_{j_n+i+1})\}_{i=1}^{L}$
end for
merge of K context sequences $\{\{(s_{j_n+i}, a_{j_n+i}, s_{j_n+i+1})\}_{i=1}^{L}\}_{n=1}^{K}$
compute latent vectors $\{Z\}_{n=1}^{K} = E_\theta(\{\{(s_{j_n+i}, a_{j_n+i}, s_{j_n+i+1})\}_{i=1}^{L}\}_{n=1}^{K})$
fix $\{Z\}_{n=1}^{K}$ as an input of D_θ
for each timestep t **do**
  current state is s_t and selected action is a_t
  predict a ensemble of next state $\{\widehat{s_{t+1}}\}_{n=1}^{K} = D_\theta(\{Z\}_{n=1}^{K}, s_t, a_t)$
  predict reward $\{\widehat{r_t}\} = f_\mu(s_t, a_t)$
end for

Appendix B. Environment Details

In this section, we present the details of the environments in our experiments.

B.1 Comparison with Model Generalization Methods

Continuing the settings in the CaDM (Lee et al., 2020) and Persim (Agarwal et al., 2021), We modify the dynamics parameters in the gym environment and assume that the reward functions and done functions are known.

B.1.1 MountainCar

In MountainCar, the goal is to drive an under-powered car to the top of a hill by taking the least number of steps.

- Observation. We observe $x(t), \dot{x}(t)$: the position and velocity of the car, respectively.
- Actions. There are three possible actions $\{0, 1, 2\}$: (0) accelerate to the left; (1) do nothing; (2) accelerate to the right.
- Reward. The reward is defined as

$$R(t) = \begin{cases} 1, & x(t) \geq 0.5 \\ -1, & \text{otherwise} \end{cases}$$

- Environment modification. We vary the gravity within the range [0.0005,0.0030]. Note that with a weaker gravity, the environment is trivially solved by directly moving to the right. On the other hand, with stronger gravity, the car must drive left and right to build up enough momentum. See Table 5 for details about the parameter ranges and the test environments.

B.1.2 CartPole

In CartPole, a pole is attached to a cart moving on a frictionless track. The goal is to prevent the pole from falling over by moving the cart to the left or to the right and to do so for as long as possible (maximum of 200 steps).

- Observation. We observe $x(t), \dot{x}(t), \theta(t), \dot{\theta}(t)$: the cart's position, its velocity, the pole's angle, and its angular velocity, respectively.
- Actions. There are two possible actions $\{0, 1\}$: (0) push to the right; (1) push to the left.
- Reward. The reward is 1 for every step taken without termination. The environment terminates when the pole angle exceeds 12 degrees or when the cart position exceeds 2.4.
- Environment modification. We vary the length of the pole and push force within the ranges $[0.3, 0.7]$ and $[5.0, 15.0]$, respectively. See Table 5 for details about the parameter ranges and the test environments.

B.1.3 HalfCheetah

In HalfCheetah, the goal is to move the cheetah as fast as possible. The cheetah's body consists of 7 links connected via 6 joints.

- Observation. We observe an 18-dimensional vector that includes the angle and angular velocity of all six joints, as well as the 3-D position and orientation of the torso. Additionally, as is done in previous studies (Agarwal et al., 2021), we append the center of mass velocity to our state vector to enable computing the reward from observations.
- Actions. The action $a(t) \in [-1, 1]^6$ represents the torque applied at the six joints.
- Reward. The reward is defined as

$$R(t) = v(t) - 0.05\|a(t)\|^2,$$

  where $v(t)$ is the center of mass velocity at time t.
- Environment modification. As in (Agarwal et al., 2021), we scale the mass of every link and the damping of every joint by factors m and d, respectively. Specifically, we vary both m and d within the range [0.5,1.5]. See Table 5 for details about the parameter ranges and the test environments.

B.1.4 SlimHumanoid

SlimHumanoid consists of 13 rigid links with 17 actuators. The goal is to move forward as fast as possible while keeping the control cost minimal.

- Observation. Observation is a 45-dimensional vector that includes angular position and velocities.
- Action. $a \in [-0.4, 0.4]^{17}$
- Reward. $r_t = 50/3 \times \dot{x}_{\text{torso},t} - 0.1\,\|a_t\|^2 + 5 \times \text{bool}\,(1.0 \leq z_{\text{torso},t} \leq 2.0)$, where $\dot{x}_{\text{torso},t}$ denotes forward velocity of the torso and z_t is the height of the torso.
- Modification. As for dynamics modification, we 1) scale the mass of every rigid link by a fixed scale factor m, and 2) scale damping of every joint by a fixed scale factor d.

B.1.5 Offline Dataset

We use the following method to collect offline data:

(i) For each gym environment, we first train a policy in an online fashion in the original task(parameters have not changed). In detail, for MountainCar and CartPole, we use DQN (Mnih et al., 2015) to train the sampling policy; for HalfCheetah, we use PPO (Schulman et al., 2017). The policies are trained to achieve rewards of approximately -40, 200, 6000, and 9000 for MountainCar, CartPole, HalfCheetah, and SlimHumanoid, respectively.

Environment	Training Range	Training Parameter	Test Parameter
MountainCar	gravity $\in [0.0005, 0.0030]$	{0.0005,0.0010,0.0018,0.0025,0.0030}	{0.0001, 0.0007, 0.0020, 0.0025, 0.0035}
CartPole(length)	length $\in [0.3, 0.7]$	{0.3,0.4,0.5,0.6,0.7}	{0.15, 0.35. 0.5, 0.65, 0.85}
CartPole(force)	force $\in [5, 15]$	{5,7.5,10,12.5,15}	{3,7,10,13,18}
HalfCheetah(mass)	relative mass $\in [0.5, 1.5]$	{0.5,0.76,1.0,1.24,1.5}	{0.3,0.6,1.0,1.4,1.7}
HalfCheetah(damping)	relative damping $\in [0.5, 1.5]$	{0.5,0.76,1.0,1.24,1.5}	{0.3,0.6,1.0,1.4,1.7}
SlimHumanoid(mass)	relative mass $\in [0.8, 1.25]$	{0.8,0.95,1.0,1.15,1.25}	{0.6,0.9,1.0,1.1,1.4}
SlimHumanoid(damping)	relative damping $\in [0.8, 1.25]$	{0.8,0.95,1.0,1.15,1.25}	{0.6,0.9,1.0,1.1,1.4}

Table 5: Environment parameters used for experiments

(*ii*) In each training task, actions are sampled according to the trained policy and a probabilistic selection parameter ϵ, which means in each state, action is selected uniformly at random with probability ϵ, respectively, and selected via the pure policy otherwise. Specifically, ϵ=0,0.2,0.4,1.0 respectively means selecting actions fully via expert policy, 20% via expert policy, 40% via expert policy, and randomly.

(*iii*) Using the action selection procedure described in (ii), 100 trajectories were collected in each training environment for the purpose of training the models. Additionally, 5 trajectories were collected in each testing task for the adaptation phase. See Table 6 for details about the reward observed for the training environments using these sampling procedures and the average reward and trajectory length achieved across all 100 times.

Environment	Data	Observed Reward					Average Trajectory Length
		environment1	environment2	environment3	environment4	environment5	
MountainCar	Pure	-102	-89	-40	-200	-186	124.4
	Pure-ϵ-20	-238	-204	-56	-189	-177	193.8
	Pure-ϵ-40	-265	-173	-72	-288	-309	222.4
	Random	-500	-500	-500	-500	-500	496.34
CartPole(length)	Pure	193	197	200	195	194	192.8
	Pure-ϵ-20	199	195	179	164	171	178.6
	Pure-ϵ-40	183	177	168	192	174	179.8
	Random	54	36	12	16	19	28.4
CartPole(force)	Pure	192	200	200	179	164	188.0
	Pure-ϵ-20	199	148	89	49	20	102.0
	Pure-ϵ-40	189	167	72	62	39	106.8
	Random	63	57	23	18	4	34.0
HalfCheetah(mass)	Pure	988.91	4032.76	6231.95	4637.34	2109.53	1000
	Pure-ϵ-20	23.49	437.82	1789.34	542.84	45.76	1000
	Pure-ϵ-40	-34.95	642.33	983.49	236.72	26.32	1000
	Random	-230.41	-346.72	-46.72	-47.28	-354.62	1000
HalfCheetah(damping)	Pure	1055.32	5281.04	6302.44	4792.63	3257.40	1000
	Pure-ϵ-20	-50.18	849.22	2378.22	1201.23	809.09	1000
	Pure-ϵ-40	-34.25	789.20	897.86	1082.36	635.41	1000
	Random	-151.47	-415.84	-178.46	-278.29	-462.91	1000
SlimHumanoid(mass)	Pure	3085.9	6056.88	8154.56	6664.6	4135.63	1000
	Pure-ϵ-20	2539.51	2160.56	5534.45	2897.27	1384.55	1000
	Pure-ϵ-40	2827.44	2522.87	3341.78	2832.21	2265.63	1000
	Random	1467.5	1566.49	2360.4	2734.82	1650.06	1000
SlimHumanoid(damping)	Pure	288.54	2778.99	8417.1	5326.89	5261.87	1000
	Pure-ϵ-20	1827.6	2367.45	4603.77	4128.65	2855.13	1000
	Pure-ϵ-40	1675.01	2749.19	3296.29	2287.76	2673.34	1000
	Random	191.78	810.97	1437.83	718.69	251.69	1000

Table 6: Observed reward and trajectory length in the four sampled datasets in each environment. Environment 1 to Environment 5 refers to the five training environments.

B.2 Comparison with Offline Meta-RL Methods

We followed the experimental environment of the previous OMRL method.

B.2.1 Environments

- **Point-Robot:** The start position is fixed at $(0, 0)$ and the goal location g is sampled from $U[-1, 1] \times U[-1, 1]$. The reward function is defined as $r_t = -\|s_t - g\|_2$, where s_t is the current position. The maximal episode steps is set to 20.
- **Ant-Dir:** The goal direction is sampled from $\theta \sim U[0, 2\pi]$. The reward function is $r_t = v_x \cos\theta + v_y \sin\theta$, where (v_x, v_y) is the horizontal velocity of the ant. The maximal episode steps is set to 200 .
- **Half-Cheetah-Vel:** The goal velocity is sampled from $v_g \sim U[0, 3]$. The reward function is $r_t = -|v_t - v_g| - \frac{1}{2}\|a_t\|_2^2$ where v_t is the current forward velocity of the agent and a_t is the action. The maximal episode steps is set to 200 .
- **Walker-Param:** Transition dynamics are varied in body mass and frictions, described by 32 parameters. Each parameter is sampled by multiplying the default value with $1.5^{\mu}, \mu \sim U[-3, 3]$. The reward function is $r_t = v_t - 10^{-3} \cdot \|a_t\|_2^2 + 1$, where v_t is the current forward velocity of the walker and a_t is the action. The episode terminates when the height of the walker is less than 0.5. The maximal episode steps is also set to 200 .
- **Hopper-Param:** Transition dynamics are varied in body mass, inertia, damping and frictions, described by 41 parameters. Each parameter is sampled by multiplying the default value with $1.5^{\mu}, \mu \sim U[-3, 3]$. The reward function is $r_t = v_t - 10^{-3} \cdot \|a_t\|_2^2$, where v_t is the current forward velocity of the hopper and a_t is the action. The maximal episode steps is set to 200.

B.2.2 Offline Dataset

For each environment, 20 training tasks and 20 testing tasks are sampled from the task distribution. On each task, we use PPO (Schulman et al., 2017) to train a single-task policy independently. The replay buffers are collected to be the offline datasets. Select 100 trajectories from the offline dataset for each task for training and 5 trajectories for adaptation.

Appendix C. Experimental Details

In this section, we list the hyperparameters in the training phases that we used to produce the experimental results when compared with model generalization methods Table 7 or OMRL methods Table 8.

Configutations	MountainCar	CartPole	HalfCheetah	SlimHumanoid
β	0.4	0.4	0.4	0.4
length of context sequence L	5	5	5	5
number of context sequence N	5	5	5	5
number of test context sequence M	5	5	5	5
Latent space dim	20	20	64	64
batch size	256	256	256	256
Training steps	2e5	2e5	2e5	2e5
decoder network width	200	200	200	200
decoder network depth	2	2	4	4
decoder activation function	Swish			
encoder network width	64	64	64	64
encoder attention head	1	1	1	1
Learning rate	$3e-4$	3e $-$ 4	$3e-4$	$3e-4$

Table 7: hyperparameters in the training phases when compared with model generalization methods.

Configutations	Point-Robot	Ant-Dir	Half-Cheetah-Vel	Walker-Param	Hopper-Param
β	0.4	0.4	0.4	0.4	0.4
length of context sequence L	5	5	5	5	5
number of context sequence N	5	5	5	5	5
number of test context sequence M	5	5	5	5	5
Latent space dim	20	20	64	64	64
batch size	256	256	256	256	256
Training steps	2e5	2e5	2e5	2e5	2e5
decoder network width	200	200	200	200	200
decoder network depth	2	2	4	4	4
decoder activation function	Swish				
encoder network width	64	64	64	64	64
encoder attention head	1	1	1	1	1
Learning rate	$3e-4$	3e $-$ 4	$3e-4$	$3e-4$	$3e-4$

Table 8: hyperparameters in the training phases when compared with OMRL methods.

Appendix D. ADDITIONAL RESULTS

We have conducted a large number of experiments to verify the effectiveness of our method, but due to space limitations, only part of the experimental results can be put down in the main text. In this section, we present the full results of our experiments. In D.1, by showing the t-SNE visualization of latent vectors, the encoder trained in the unsupervised way we proposed can indeed extract effective information about dynamic parameters from the context. D.2, D.3, respectively show that our PLDMM has a smaller prediction error and higher average reward than previous model generalization methods.

D.1 Visualization of Latent Factors

In this part, we show the t-SNE visualization of context latent vectors generated by trained and untrained encoders in experimental environments.

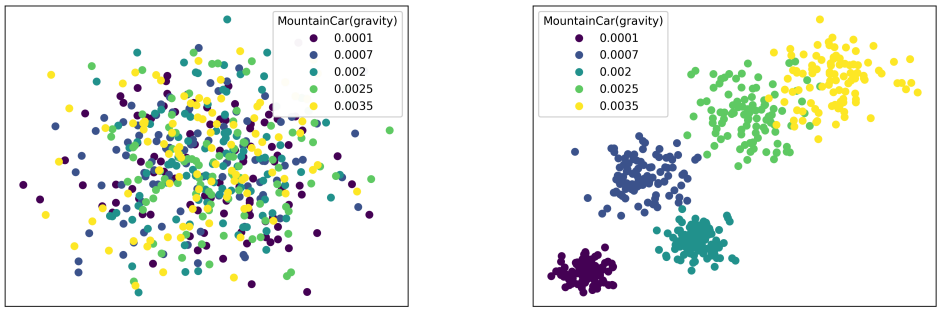


Figure 5: t-SNE visualization of context latent vectors extracted from contexts collected in various environments in MountainCar(gravity). Embedded points from environments with the same parameter have the same colour.

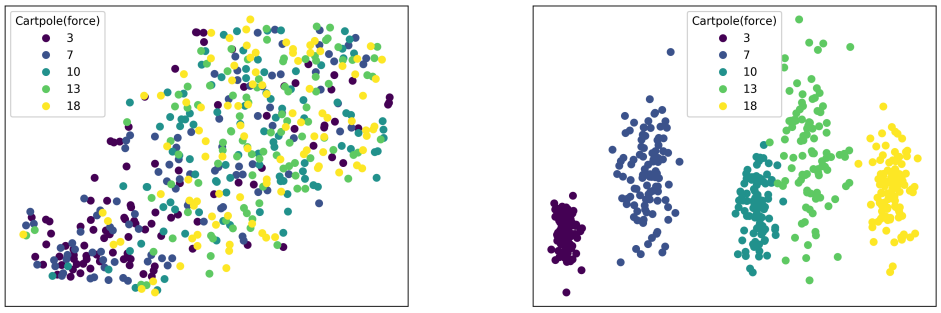


Figure 6: t-SNE visualization of context latent vectors extracted from contexts collected in various environments in CartPole(force). Embedded points from environments with the same parameter have the same colour.

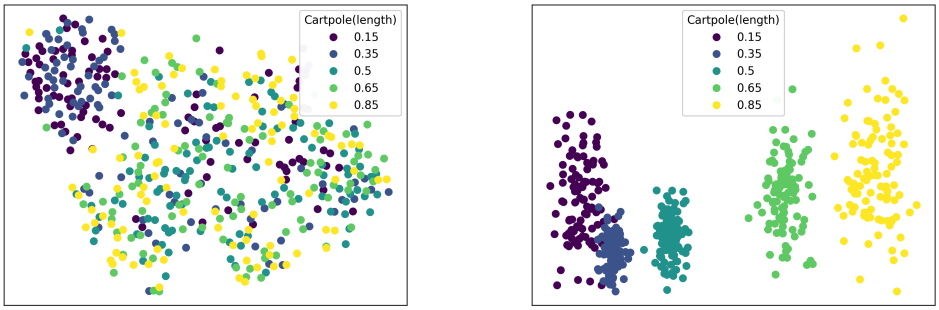


Figure 7: t-SNE visualization of context latent vectors extracted from contexts collected in various environments in CartPole(length). Embedded points from environments with the same parameter have the same colour.

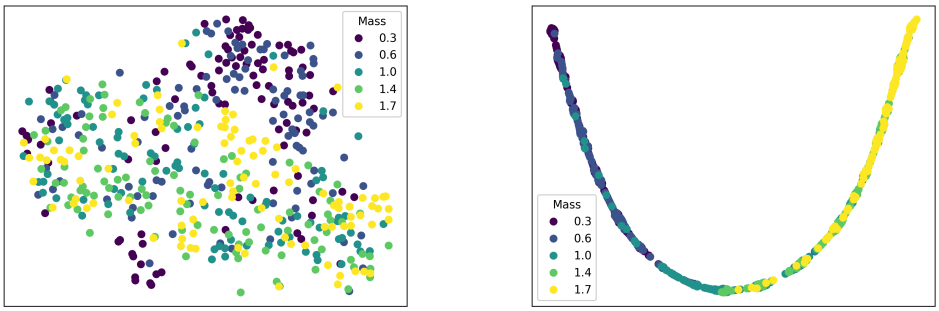


Figure 8: t-SNE visualization of context latent vectors extracted from contexts collected in various environments in Halfcheetah(Mass). Embedded points from environments with the same parameter have the same colour.

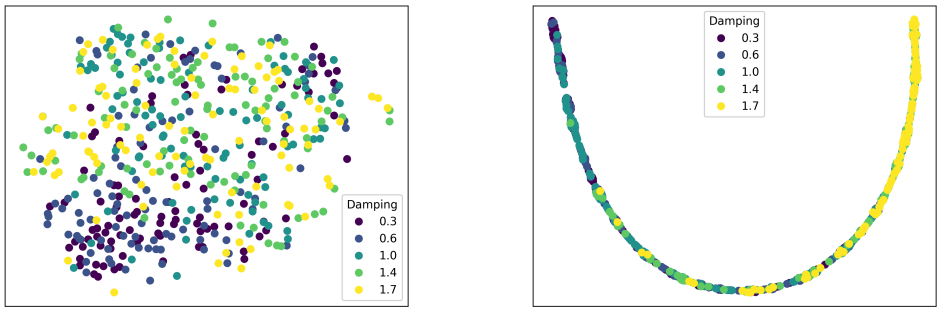


Figure 9: t-SNE visualization of context latent vectors extracted from contexts collected in various environments in Halfcheetah(Damping). Embedded points from environments with the same parameter have the same colour.

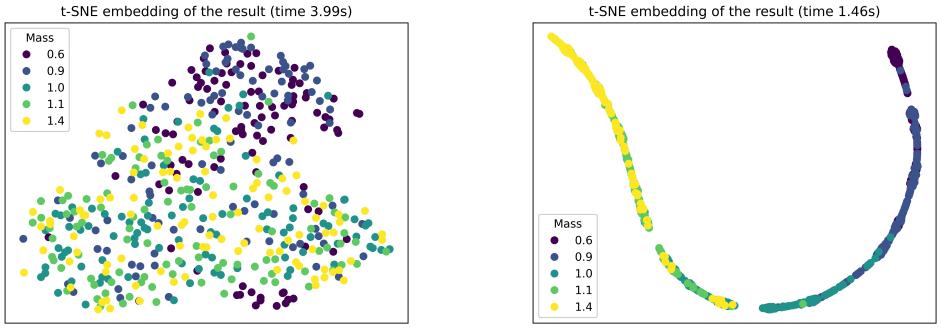


Figure 10: t-SNE visualization of context latent vectors extracted from contexts collected in various environments in SlimHumanoid(Mass). Embedded points from environments with the same parameter have the same colour.

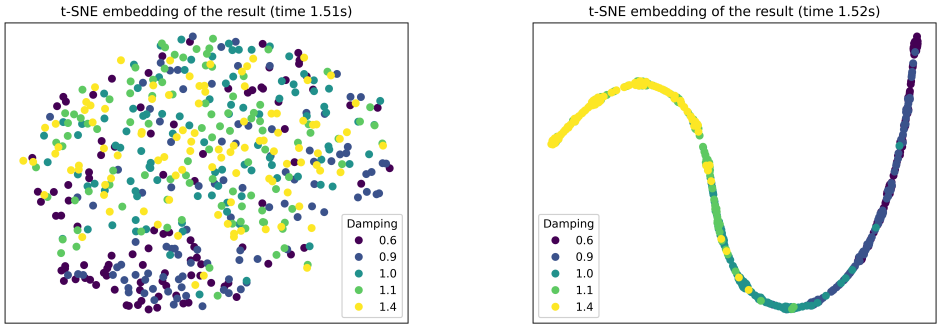


Figure 11: t-SNE visualization of context latent vectors extracted from contexts collected in various environments in SlimHumanoid(Damping). Embedded points from environments with the same parameter have the same colour.

D.2 Detailed Prediction Error Results

Data	Method	Dynamic1(0.0001)	Dynamic2(0.0007)	Dynamic3(0.0020)	Dynamic4(0.0025)	Dynamic5(0.0035)
Pure	Persim	1.318(±0.142)	1.147(±0.208)	0.902(±0.185)	1.001(±0.215)	1.076(±0.248)
	Persim-M	3.101(±0.135)	3.237(±0.144)	3.210(±0.146)	2.749(±0.176)	4.484(±0.431)
	Vanilla CaDM	4.296(±1.183)	3.530(±1.235)	3.698(±1.007)	4.201(±1.399)	4.903(±1.286)
	PE-TS CaDM	5.513(±1.495)	3.093(±1.119)	2.034(±0.705)	1.141(±0.518)	3.386(±0.987)
	OURS	0.962(±0.328)	0.338(±0.023)	0.752(±0.032)	0.675(±0.039)	0.493(±0.033)
Pure-ϵ-20	Persim	1.11(±0.134)	1.081(±0.202)	1.065(±0.15)	0.896(±0.201)	0.864(±0.216)
	Persim-M	2.659(±0.129)	2.352(±0.097)	2.396(±0.137)	2.572(±0.153)	4.635(±0.443)
	Vanilla CaDM	3.981(±1.162)	3.383(±1.173)	3.815(±1.182)	4.296(±1.317)	3.925(±1.137)
	PE-TS CaDM	4.831(±1.27)	2.933(±0.938)	1.366(±0.5)	1.346(±0.603)	3.568(±1.303)
	OURS	0.95(±0.409)	0.497(±0.02)	0.755(±0.042)	0.502(±0.027)	0.638(±0.038)
Pure-ϵ-40	Persim	1.36(±0.157)	1.603(±0.247)	1.372(±0.171)	0.902(±0.202)	0.856(±0.138)
	Persim-M	2.866(±0.108)	3.104(±0.152)	2.305(±0.12)	3.337(±0.164)	4.713(±0.404)
	Vanilla CaDM	4.427(±1.36)	5.003(±1.477)	4.306(±1.206)	3.993(±1.122)	4.233(±1.144)
	PE-TS CaDM	5.533(±1.737)	2.726(±0.831)	1.958(±0.858)	0.824(±0.517)	3.01(±1.069)
	OURS	0.887(±0.359)	0.478(±0.02)	0.46(±0.028)	0.532(±0.035)	0.582(±0.038)
Random	Persim	0.968(±0.127)	0.979(±0.13)	0.929(±0.19)	1.097(±0.212)	0.776(±0.171)
	Persim-M	3.333(±0.142)	2.755(±0.124)	2.849(±0.162)	3.806(±0.159)	5.098(±0.471)
	Vanilla CaDM	4.52(±1.148)	2.603(±0.922)	3.876(±1.442)	4.876(±1.218)	4.281(±1.368)
	PE-TS CaDM	3.985(±1.375)	2.035(±0.742)	2.003(±0.76)	1.093(±0.613)	3.063(±0.914)
	OURS	0.773(±0.304)	0.59(±0.016)	0.629(±0.024)	0.661(±0.041)	0.691(±0.033)

Table 9: Prediction Error: MountainCar(gravity)

Data	Method	Dynamic1(3)	Dynamic2(7)	Dynamic3(10)	Dynamic4(13)	Dynamic5(18)
Pure	Persim	0.882(±0.136)	1.538(±0.187)	1.065(±0.177)	1.421(±0.233)	0.595(±0.166)
	Persim-M	3.758(±0.159)	2.502(±0.123)	3.133(±0.185)	2.743(±0.17)	5.031(±0.538)
	Vanilla CaDM	5.719(±1.246)	3.718(±0.696)	1.808(±0.762)	3.246(±0.892)	3.266(±1.098)
	PE-TS CaDM	4.688(±0.621)	2.885(±0.372)	1.721(±0.212)	2.121(±0.664)	2.402(±0.415)
	OURS	0.819(±0.273)	0.442(±0.021)	0.517(±0.031)	0.692(±0.036)	0.614(±0.048)
Pure-ϵ-20	Persim	1.429(±0.203)	1.031(±0.131)	1.289(±0.176)	1.174(±0.188)	1.103(±0.191)
	Persim-M	2.942(±0.114)	2.851(±0.115)	2.328(±0.12)	2.925(±0.166)	4.453(±0.443)
	Vanilla CaDM	5.88(±1.101)	3.863(±1.006)	2.047(±0.78)	3.031(±0.847)	4.292(±1.418)
	PE-TS CaDM	4.216(±0.548)	2.819(±0.37)	1.931(±0.219)	1.903(±0.536)	2.943(±0.502)
	OURS	0.787(±0.245)	0.522(±0.021)	0.72(±0.034)	0.615(±0.035)	0.718(±0.041)
Pure-ϵ-40	Persim	1.434(±0.18)	0.968(±0.149)	0.777(±0.174)	1.444(±0.193)	1.006(±0.17)
	Persim-M	3.393(±0.122)	3.088(±0.117)	2.613(±0.112)	3.249(±0.184)	3.579(±0.333)
	Vanilla CaDM	5.782(±1.136)	4.026(±0.954)	1.824(±0.788)	3.029(±0.675)	4.267(±1.398)
	PE-TS CaDM	4.173(±0.581)	3.294(±0.332)	2.013(±0.199)	1.887(±0.7)	2.685(±0.594)
	OURS	0.784(±0.303)	0.526(±0.018)	0.711(±0.035)	0.613(±0.04)	0.716(±0.033)
Random	Persim	1.597(±0.162)	1.079(±0.144)	1.211(±0.183)	0.944(±0.136)	1.111(±0.2)
	Persim-M	2.918(±0.117)	2.78(±0.118)	2.769(±0.165)	2.783(±0.144)	3.945(±0.377)
	Vanilla CaDM	6.757(±1.215)	3.671(±0.981)	1.788(±0.707)	3.239(±0.958)	4.156(±1.053)
	PE-TS CaDM	4.035(±0.546)	2.756(±0.313)	2.001(±0.194)	1.892(±0.564)	3.224(±0.723)
	OURS	0.711(±0.307)	0.622(±0.021)	0.645(±0.03)	0.603(±0.047)	0.817(±0.058)

Table 10: Prediction Error: CartPole(force)

Data	Method	Dynamic1(0.15)	Dynamic2(0.35)	Dynamic3(0.5)	Dynamic4(0.65)	Dynamic5(0.85)
Pure	Persim	1.921(±0.187)	1.176(±0.159)	1.138(±0.156)	0.968(±0.162)	0.735(±0.188)
	Persim-M	49.355(±37.759)	12.997(±9.822)	4.157(±2.282)	4.945(±2.484)	48.352(±27.386)
	Vanilla CaDM	5.847(±1.096)	4.137(±0.897)	1.831(±0.613)	2.724(±0.66)	4.102(±1.202)
	PE-TS CaDM	5.339(±1.277)	3.989(±0.726)	2.213(±0.864)	2.965(±0.754)	3.977(±1.095)
	OURS	0.829(±0.266)	0.651(±0.019)	0.393(±0.022)	0.553(±0.036)	0.955(±0.059)
Pure-ϵ-20	Persim	1.552(±0.171)	0.848(±0.127)	1.314(±0.172)	0.854(±0.164)	0.625(±0.138)
	Persim-M	3.146(±0.121)	2.578(±0.119)	2.873(±0.168)	2.59(±0.131)	3.668(±0.381)
	Vanilla CaDM	8.848(±2.324)	3.07(±0.568)	1.82(±0.805)	2.953(±0.76)	3.872(±1.015)
	PE-TS CaDM	5.732(±1.431)	4.427(±1.024)	1.811(±0.683)	2.859(±0.63)	4.25(±0.957)
	OURS	0.734(±0.335)	0.708(±0.018)	0.664(±0.031)	0.574(±0.047)	0.592(±0.046)
Pure-ϵ-40	Persim	1.759(±0.245)	0.428(±0.09)	0.824(±0.148)	0.987(±0.151)	1.477(±0.213)
	Persim-M	3.642(±0.126)	2.668(±0.113)	3.346(±0.202)	3.089(±0.158)	3.974(±0.41)
	Vanilla CaDM	5.278(±1.343)	4.317(±1.265)	3.556(±1.082)	4.192(±0.965)	4.465(±1.418)
	PE-TS CaDM	3.927(±1.304)	4.368(±1.184)	3.391(±0.954)	3.939(±1.256)	4.739(±1.744)
	OURS	0.807(±0.283)	0.681(±0.022)	0.64(±0.029)	0.643(±0.035)	1.037(±0.061)
Random	Persim	1.374(±0.176)	1.172(±0.143)	1.612(±0.225)	1.092(±0.197)	1.113(±0.21)
	Persim-M	2.66(±0.125)	2.193(±0.1)	3.332(±0.187)	2.427(±0.153)	3.561(±0.402)
	Vanilla CaDM	4.303(±1.372)	4.623(±1.324)	3.251(±0.959)	3.763(±1.059)	4.272(±1.519)
	PE-TS CaDM	3.987(±1.03)	5.007(±1.402)	3.463(±0.937)	4.072(±1.367)	3.979(±1.292)
	OURS	0.568(±0.207)	0.54(±0.021)	0.575(±0.03)	0.587(±0.046)	0.724(±0.051)

Table 11: Prediction Error: CartPole(length)

Data	Method	Dynamic1(0.3)	Dynamic2(0.6)	Dynamic3(1.0)	Dynamic4(1.4)	Dynamic5(1.7)
Pure	Persim	4.02(±2.305)	3.698(±2.311)	3.115(±2.794)	3.484(±3.032)	3.635(±2.993)
	Persim-M	64.975(±60.354)	11.002(±8.315)	3.576(±2.074)	4.894(±4.879)	58.088(±30.926)
	Vanilla CaDM	6.247(±1.596)	3.878(±0.895)	1.795(±0.646)	2.904(±0.815)	3.88(±1.013)
	PE-TS CaDM	5.321(±1.74)	3.14(±0.911)	1.827(±0.849)	1.191(±0.537)	3.109(±0.856)
	OURS	3.883(±1.079)	2.499(±1.163)	1.721(±0.905)	1.182(±0.288)	3.781(±0.682)
Pure-ϵ-20	Persim	1.155(±0.167)	1.181(±0.133)	1.157(±0.161)	1.123(±0.189)	1.108(±0.205)
	Persim-M	5.247(±4.282)	2.989(±0.163)	5.827(±5.263)	13.694(±6.255)	46.063(±29.753)
	Vanilla CaDM	6.017(±1.022)	3.587(±1.13)	2.058(±0.922)	2.259(±0.837)	3.438(±1.23)
	PE-TS CaDM	4.825(±1.959)	2.918(±1.644)	1.976(±1.108)	2.059(±0.736)	2.967(±1.0)
	OURS	1.122(±0.043)	1.121(±0.756)	1.169(±0.768)	0.981(±0.059)	1.015(±0.038)
Pure-ϵ-40	Persim	1.073(±0.145)	1.131(±0.169)	1.081(±0.164)	1.003(±0.153)	0.969(±0.163)
	Persim-M	2.726(±0.133)	2.467(±0.689)	2.013(±0.128)	2.089(±0.108)	2.176(±0.149)
	Vanilla CaDM	5.965(±2.23)	3.613(±1.21)	2.147(±1.433)	2.552(±0.88)	3.69(±1.163)
	PE-TS CaDM	5.046(±1.53)	3.027(±1.399)	1.8(±0.767)	1.926(±1.125)	3.043(±0.957)
	OURS	1.008(±0.913)	0.754(±0.545)	0.672(±0.044)	0.744(±0.035)	0.793(±0.063)
Random	Persim	1.024(±0.124)	1.053(±0.151)	0.99(±0.177)	0.93(±0.206)	0.889(±0.207)
	Persim-M	3.277(±0.145)	2.736(±0.13)	2.74(±0.119)	3.121(±0.136)	4.184(±0.35)
	Vanilla CaDM	4.465(±1.171)	4.127(±1.387)	3.833(±1.308)	4.001(±1.087)	4.096(±1.523)
	PE-TS CaDM	4.176(±0.737)	2.727(±0.362)	1.967(±0.234)	2.016(±0.662)	2.761(±0.604)
	OURS	0.847(±0.317)	0.537(±0.017)	0.591(±0.027)	0.644(±0.044)	0.71(±0.036)

Table 12: Prediction Error: HalfCheetah(Mass Scale)

Data	Method	Dynamic1(0.3)	Dynamic2(0.6)	Dynamic3(1.0)	Dynamic4(1.4)	Dynamic5(1.7)
Pure	Persim	4.188(±2.748)	3.744(±2.214)	3.29(±2.96)	3.868(±3.233)	3.525(±3.127)
	Persim-M	70.31(±67.367)	9.651(±7.697)	3.623(±1.598)	5.325(±2.888)	70.749(±53.026)
	Vanilla CaDM	5.959(±1.069)	3.885(±0.976)	1.956(±0.676)	2.928(±0.713)	4.086(±1.363)
	PE-TS CaDM	6.428(±2.317)	3.522(±1.311)	1.788(±0.625)	1.098(±0.459)	3.614(±1.202)
	OURS	13.749(±1.419)	10.244(±1.031)	8.283(±0.944)	7.598(±0.279)	7.402(±0.422)
Pure-ϵ-20	Persim	1.164(±0.14)	1.249(±0.153)	1.032(±0.148)	1.18(±0.202)	1.157(±0.196)
	Persim-M	5.917(±3.589)	3.397(±0.165)	5.944(±3.008)	12.78(±6.324)	42.341(±22.585)
	Vanilla CaDM	5.607(±1.253)	3.729(±1.276)	2.176(±1.028)	2.314(±0.778)	3.006(±1.309)
	PE-TS CaDM	4.495(±1.778)	2.765(±1.344)	1.898(±1.009)	1.631(±0.565)	3.042(±1.147)
	OURS	1.051(±0.032)	1.178(±0.995)	1.186(±0.745)	0.985(±0.049)	1.063(±0.034)
Pure-ϵ-40	Persim	0.86(±0.103)	0.988(±0.139)	1.177(±0.2)	0.997(±0.199)	0.954(±0.178)
	Persim-M	2.619(±0.144)	2.611(±0.756)	2.139(±0.12)	2.332(±0.126)	2.341(±0.14)
	Vanilla CaDM	5.393(±1.651)	3.643(±1.309)	2.063(±1.813)	2.588(±0.893)	3.78(±1.443)
	PE-TS CaDM	5.441(±1.807)	2.661(±1.262)	1.686(±0.738)	1.794(±1.068)	2.809(±1.169)
	OURS	0.956(±0.848)	0.76(±0.628)	0.676(±0.038)	0.744(±0.028)	0.791(±0.084)
Random	Persim	1.097(±0.144)	0.949(±0.134)	1.0(±0.182)	0.922(±0.212)	0.881(±0.178)
	Persim-M	2.935(±0.141)	2.637(±0.11)	2.88(±0.131)	2.847(±0.122)	4.363(±0.419)
	Vanilla CaDM	4.375(±1.414)	3.955(±1.198)	4.32(±1.277)	4.335(±1.33)	4.008(±1.018)
	PE-TS CaDM	2.954(±0.529)	2.9(±0.38)	1.836(±0.189)	2.072(±0.618)	3.108(±0.595)
	OURS	0.834(±0.376)	0.407(±0.024)	0.534(±0.029)	0.731(±0.042)	0.82(±0.052)

Table 13: Prediction Error: HalfCheetah(Damping Scale)

Data	Method	Dynamic1(0.3)	Dynamic2(0.6)	Dynamic3(1.0)	Dynamic4(1.4)	Dynamic5(1.7)
Pure	Persim	4.06(±2.966)	5.007(±3.886)	3.627(±1.793)	3.625(±2.823)	3.648(±3.069)
	Persim-M	47.042(±30.777)	11.158(±11.113)	5.402(±1.6)	5.103(±2.831)	41.025(±27.582)
	Vanilla CaDM	6.691(±1.268)	5.356(±0.951)	0.834(±0.552)	2.156(±0.683)	4.483(±1.075)
	PE-TS CaDM	4.316(±1.608)	3.447(±0.97)	3.558(±0.608)	1.819(±0.407)	4.267(±1.021)
	OURS	12.27(±1.093)	11.118(±1.029)	9.177(±0.655)	8.599(±0.389)	5.858(±0.338)
Pure-ϵ-20	Persim	0.947(±0.199)	0.537(±0.118)	2.613(±0.179)	2.039(±0.173)	1.543(±0.169)
	Persim-M	5.748(±3.552)	4.95(±0.129)	6.551(±4.216)	14.965(±8.406)	52.794(±27.678)
	Vanilla CaDM	5.069(±1.045)	3.531(±1.08)	2.294(±0.736)	2.708(±0.948)	4.527(±1.055)
	PE-TS CaDM	3.861(±2.225)	3.202(±1.257)	1.921(±1.059)	3.909(±0.669)	4.291(±1.18)
	OURS	0.606(±0.035)	0.617(±0.544)	2.917(±0.814)	0.14(±0.045)	0.94(±0.037)
Pure-ϵ-40	Persim	2.076(±0.129)	1.339(±0.183)	2.774(±0.144)	2.065(±0.127)	2.355(±0.258)
	Persim-M	2.745(±0.142)	3.235(±0.921)	3.169(±0.097)	3.14(±0.109)	3.568(±0.14)
	Vanilla CaDM	5.269(±1.546)	5.154(±1.131)	3.478(±1.516)	4.152(±1.273)	4.218(±1.227)
	PE-TS CaDM	4.474(±1.299)	4.567(±1.545)	1.849(±0.994)	3.338(±0.876)	2.087(±1.161)
	OURS	2.279(±1.092)	0.524(±0.291)	2.202(±0.025)	1.203(±0.03)	1.348(±0.07)
Random	Persim	0.767(±0.146)	1.162(±0.144)	1.819(±0.189)	0.652(±0.205)	0.614(±0.195)
	Persim-M	4.424(±0.115)	3.008(±0.1)	2.522(±0.16)	3.986(±0.119)	4.638(±0.403)
	Vanilla CaDM	4.831(±1.236)	3.687(±1.699)	3.75(±1.234)	5.211(±1.01)	5.008(±1.236)
	PE-TS CaDM	3.931(±0.561)	2.059(±0.333)	2.511(±0.152)	3.872(±0.728)	2.861(±0.632)
	OURS	2.39(±0.332)	0.487(±0.02)	1.982(±0.033)	0.332(±0.047)	2.511(±0.058)

Table 14: Prediction Error: SlimHumanoid(Mass Scale)

Data	Method	Dynamic1(0.6)	Dynamic2(0.9)	Dynamic3(1.0)	Dynamic4(1.1)	Dynamic5(1.4)
Pure	Persim	3.347(±2.236)	3.209(±2.68)	2.361(±2.243)	3.178(±1.476)	4.776(±3.165)
	Persim-M	87.276(±71.141)	9.851(±8.859)	4.037(±1.48)	4.556(±2.948)	77.969(±46.563)
	Vanilla CaDM	6.192(±1.17)	5.213(±0.841)	1.966(±0.575)	2.447(±0.747)	3.442(±0.788)
	PE-TS CaDM	6.33(±2.232)	2.998(±1.256)	1.119(±0.753)	0.17(±0.136)	3.097(±1.295)
	OURS	12.278(±1.404)	13.68(±1.066)	9.184(±0.936)	9.176(±0.295)	6.752(±0.327)
Pure-ϵ-20	Persim	0.21(±0.143)	0.705(±0.115)	2.186(±0.159)	0.723(±0.224)	2.592(±0.136)
	Persim-M	5.676(±3.368)	4.385(±0.222)	6.369(±5.209)	14.289(±9.714)	43.884(±31.39)
	Vanilla CaDM	5.096(±1.14)	3.777(±1.386)	1.519(±0.77)	4.045(±0.64)	3.991(±1.07)
	PE-TS CaDM	6.065(±2.317)	2.237(±1.181)	1.161(±0.8)	0.727(±0.565)	2.355(±1.409)
	OURS	0.737(±0.035)	1.216(±0.765)	1.397(±0.942)	2.197(±0.06)	2.272(±0.049)
Pure-ϵ-40	Persim	2.603(±0.098)	2.725(±0.135)	0.753(±0.185)	2.818(±0.185)	1.945(±0.159)
	Persim-M	2.091(±0.175)	3.599(±0.821)	2.882(±0.164)	2.099(±0.159)	2.513(±0.131)
	Vanilla CaDM	4.591(±1.53)	4.876(±1.499)	2.709(±1.716)	3.958(±1.003)	3.73(±1.559)
	PE-TS CaDM	4.928(±1.492)	2.847(±1.317)	2.791(±0.709)	3.143(±1.111)	2.88(±0.841)
	OURS	0.717(±0.559)	1.552(±0.706)	2.074(±0.043)	1.614(±0.031)	0.974(±0.077)
Random	Persim	2.306(±0.155)	0.292(±0.142)	0.24(±0.138)	1.358(±0.253)	1.064(±0.156)
	Persim-M	3.616(±0.139)	1.742(±0.127)	3.297(±0.184)	2.303(±0.164)	5.525(±0.42)
	Vanilla CaDM	4.943(±1.086)	3.743(±1.311)	3.641(±1.059)	3.612(±1.198)	4.214(±1.406)
	PE-TS CaDM	2.694(±0.444)	3.242(±0.432)	3.362(±0.232)	1.43(±0.583)	3.014(±0.835)
	OURS	0.762(±0.526)	2.203(±0.022)	1.292(±0.033)	0.783(±0.055)	1.647(±0.034)

Table 15: Prediction Error: SlimHumanoid(Damping Scale)

D.3 Detailed Average Reward Results

Data	Method	Dynamic1(0.0001)	Dynamic2(0.0007)	Dynamic3(0.0020)	Dynamic4(0.0025)	Dynamic5(0.0035)
Pure	Persim	-55.0(±8.6)	-139.3(±4.5)	-156.6(±6.8)	-246.4(±2.8)	-286.4(±11.9)
	Persim-M	-80.7(±9.0)	-138.5(±4.9)	-181.0(±7.3)	-300.9(±3.1)	-343.9(±14.3)
	Vanilla CaDM	-93.7(±9.3)	-237.0(±3.1)	-239.2(±5.1)	-383.5(±3.1)	-377.5(±12.9)
	PE-TS CaDM	-71.8(±7.5)	-176.6(±4.6)	-194.9(±6.1)	-296.9(±2.9)	-331.4(±10.7)
	OURS	-21.2(±7.9)	-91.3(±4.4)	-134.9(±5.9)	-234.5(±2.8)	-252.8(±13.2)
	OURS + PPO	-54.5(±6.7)	-88.1(±3.1)	-120.9(±5.7)	-198.0(±2.9)	-80.0(±9.9)
Pure-ϵ-20	Persim	-52.8(±8.3)	-132.7(±4.5)	-135.5(±8.4)	-242.4(±2.8)	-285.5(±12.5)
	Persim-M	-110.1(±11.4)	-169.5(±3.6)	-247.4(±7.0)	-314.8(±3.0)	-500.0(±0.0)
	Vanilla CaDM	-157.7(±10.5)	-174.0(±3.4)	-254.7(±8.4)	-408.1(±3.3)	-500.0(±0.0)
	PE-TS CaDM	-72.1(±8.0)	-170.2(±3.8)	-185.3(±4.8)	-291.5(±2.8)	-333.7(±10.2)
	OURS	-16.9(±7.7)	-83.8(±5.2)	-95.1(±4.9)	-153.2(±4.0)	-183.2(±14.2)
	OURS + PPO	-53.4(±7.8)	-83.3(±4.1)	-87.6(±8.9)	-184.5(±3.9)	-75.4(±8.0)
Pure-ϵ-40	Persim	-46.3(±7.2)	-123.2(±4.3)	-131.4(±8.3)	-210.1(±3.3)	-232.9(±11.8)
	Persim-M	-73.1(±12.0)	-119.3(±5.5)	-178.3(±7.7)	-279.1(±2.7)	-350.6(±12.7)
	Vanilla CaDM	-152.0(±10.6)	-200.4(±4.3)	-329.8(±7.9)	-495.8(±4.1)	-500.0(±0.0)
	PE-TS CaDM	-65.8(±7.5)	-146.9(±4.3)	-156.2(±4.2)	-270.4(±2.3)	-303.5(±11.2)
	OURS	-17.7(±9.6)	-87.5(±4.9)	-116.9(±6.6)	-167.3(±3.8)	-197.5(±7.8)
	OURS + PPO	-52.1(±9.0)	-81.9(±3.9)	-79.1(±7.9)	-178.0(±3.8)	-72.2(±7.8)
Random	Persim	-37.0(±5.3)	-118.9(±3.4)	-122.1(±7.3)	-174.9(±2.5)	-228.9(±12.3)
	Persim-M	-65.8(±9.4)	-111.9(±3.9)	-150.0(±4.8)	-276.4(±3.7)	-329.1(±11.1)
	Vanilla CaDM	-148.1(±14.1)	-190.5(±4.2)	-372.9(±7.1)	-500.0(±0.0)	-500.0(±0.0)
	PE-TS CaDM	-53.6(±11.0)	-144.3(±3.9)	-144.6(±5.1)	-264.3(±2.9)	-305.4(±13.0)
	OURS	-16.7(±9.9)	-87.4(±4.9)	-114.3(±7.0)	-158.7(±3.3)	-166.0(±9.0)
	OURS + PPO	-49.7(±8.0)	-78.3(±3.8)	-76.8(±9.6)	-180.1(±3.6)	-64.8(±8.5)

Table 16: Average Reward: MountainCar(gravity)

Data	Method	Dynamic1(3)	Dynamic2(7)	Dynamic3(10)	Dynamic4(13)	Dynamic5(18)
Pure	Persim	156.9(±5.8)	199.1(±0.9)	198.7(±0.0)	133.1(±6.2)	106.4(±13.1)
	Persim-M	152.9(±5.3)	174.4(±11.4)	190.8(±7.3)	124.7(±5.5)	100.0(±8.8)
	Vanilla CaDM	133.4(±5.1)	179.2(±11.4)	197.0(±1.8)	120.9(±6.6)	95.8(±14.2)
	PE-TS CaDM	149.1(±7.1)	186.6(±8.2)	179.7(±5.9)	127.2(±8.7)	104.4(±9.4)
	OURS	168.1(±7.0)	200.0(±0.0)	200.0(±0.0)	138.0(±7.5)	106.6(±10.6)
	OURS + PPO	156.6(±5.2)	200.0(±0.0)	200.0(±0.0)	153.6(±6.7)	119.9(±15.3)
Pure-ϵ-20	Persim	163.7(±6.2)	200.0(±0.0)	200.0(±0.0)	141.3(±6.3)	108.7(±10.6)
	Persim-M	158.7(±7.9)	172.6(±8.7)	191.7(±5.7)	125.0(±6.0)	108.8(±14.9)
	Vanilla CaDM	141.3(±3.9)	183.4(±9.3)	200.0(±0.0)	123.9(±6.8)	101.5(±15.3)
	PE-TS CaDM	164.3(±5.9)	200.0(±0.0)	188.9(±7.5)	135.7(±6.3)	115.6(±11.0)
	OURS	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)
	OURS + PPO	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)
Pure-ϵ-40	Persim	158.9(±5.3)	200.0(±0.0)	200.0(±0.0)	141.5(±5.7)	108.5(±10.5)
	Persim-M	156.0(±5.6)	181.0(±7.5)	200.0(±0.0)	124.5(±7.7)	117.0(±14.7)
	Vanilla CaDM	137.1(±5.9)	191.4(±6.8)	200.0(±0.0)	141.9(±4.4)	100.8(±12.7)
	PE-TS CaDM	162.1(±6.8)	194.5(±3.8)	187.4(±7.4)	140.4(±6.0)	107.3(±12.7)
	OURS	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)
	OURS + PPO	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)
Random	Persim	164.4(±5.4)	200.0(±0.0)	200.0(±0.0)	131.8(±7.4)	124.7(±12.4)
	Persim-M	155.0(±7.2)	200.0(±0.0)	200.0(±0.0)	127.6(±8.2)	107.0(±9.7)
	Vanilla CaDM	131.7(±5.8)	195.0(±3.4)	200.0(±0.0)	125.1(±5.4)	104.9(±16.9)
	PE-TS CaDM	159.5(±4.9)	196.1(±2.1)	189.2(±7.1)	131.8(±7.8)	108.6(±9.9)
	OURS	177.7(±7.2)	200.0(±0.0)	200.0(±0.0)	146.6(±8.6)	118.5(±15.5)
	OURS + PPO	164.2(±6.5)	200.0(±0.0)	200.0(±0.0)	173.8(±8.4)	128.9(±10.7)

Table 17: Average Reward: CartPole(force)

Data	Method	Dynamic1(0.15)	Dynamic2(0.35)	Dynamic3(0.5)	Dynamic4(0.65)	Dynamic5(0.85)
Pure	Persim	164.8(±5.0)	200.0(±0.0)	200.0(±0.0)	158.5(±7.3)	130.6(±15.5)
	Persim-M	166.8(±7.5)	200.0(±0.0)	200.0(±0.0)	149.4(±5.3)	105.0(±9.2)
	Vanilla CaDM	160.8(±5.2)	200.0(±0.0)	200.0(±0.0)	146.2(±7.6)	121.9(±9.6)
	PE-TS CaDM	189.9(±7.3)	200.0(±0.0)	200.0(±0.0)	141.8(±7.8)	106.9(±12.8)
	OURS	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)
	OURS + PPO	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)
Pure-ϵ-20	Persim	184.8(±6.9)	200.0(±0.0)	200.0(±0.0)	137.1(±6.0)	153.6(±11.4)
	Persim-M	160.7(±6.5)	200.0(±0.0)	200.0(±0.0)	171.2(±8.9)	128.4(±8.3)
	Vanilla CaDM	162.7(±6.0)	200.0(±0.0)	200.0(±0.0)	158.9(±8.7)	137.2(±9.9)
	PE-TS CaDM	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	178.2(±7.3)	119.6(±10.3)
	OURS	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)
	OURS + PPO	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)
Pure-ϵ-40	Persim	179.4(±7.2)	196.2(±1.9)	200.0(±0.0)	147.6(±5.9)	135.9(±13.1)
	Persim-M	181.3(±6.1)	198.0(±0.5)	200.0(±0.0)	167.2(±6.5)	117.1(±10.5)
	Vanilla CaDM	193.6(±6.2)	194.0(±4.4)	200.0(±0.0)	192.6(±5.4)	143.7(±9.4)
	PE-TS CaDM	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	165.1(±8.5)	118.6(±12.4)
	OURS	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)
	OURS + PPO	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)
Random	Persim	175.4(±6.3)	200.0(±0.0)	200.0(±0.0)	149.4(±6.1)	128.4(±13.8)
	Persim-M	177.6(±5.7)	182.9(±12.2)	188.0(±10.6)	160.8(±7.4)	139.2(±16.4)
	Vanilla CaDM	190.7(±6.4)	199.3(±1.4)	197.1(±1.3)	171.8(±6.2)	133.8(±13.6)
	PE-TS CaDM	181.3(±5.3)	200.0(±0.0)	200.0(±0.0)	189.1(±7.2)	145.7(±13.2)
	OURS	179.6(±5.6)	200.0(±0.0)	200.0(±0.0)	185.1(±8.4)	147.9(±19.5)
	OURS + PPO	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)	200.0(±0.0)

Table 18: Average Reward: CartPole(length)

Data	Method	Dynamic1(0.3)	Dynamic2(0.6)	Dynamic3(1.0)	Dynamic4(1.4)	Dynamic5(1.7)
Pure	Persim	54.7(±68.3)	136.9(±52.3)	-161.7(±11.1)	-48.9(±7.2)	-108.5(±13.3)
	Persim-M	-44.9(±10.6)	-93.3(±21.3)	-152.4(±1.7)	-145.6(±2.1)	-104.2(±2.2)
	Vanilla CaDM	-1.9(±74.5)	-47.4(±34.9)	-1128.2(±79.7)	-66.2(±5.9)	-333.3(±8.3)
	PE-TS CaDM	-224.3(±26.7)	16.6(±28.9)	-163.9(±10.4)	-300.3(±2.6)	-185.9(±17.9)
	OURS	289.3(±27.0)	364.6(±99.9)	55.4(±4.2)	19.9(±8.0)	-1.8(±19.5)
	OURS + PPO	205.4(±182.3)	171.6(±42.1)	76.1(±2.7)	72.3(±3.8)	3.7(±5.3)
Pure-ϵ-20	Persim	302.2(±229.2)	277.3(±20.0)	821.7(±276.2)	803.5(±31.8)	2125.8(±12.9)
	Persim-M	328.2(±33.6)	-34.2(±10.6)	626.2(±12.3)	438.2(±13.4)	357.5(±16.2)
	Vanilla CaDM	208.7(±272.2)	66.0(±37.2)	540.9(±362.9)	468.6(±60.7)	1952.7(±9.0)
	PE-TS CaDM	234.0(±303.4)	-253.0(±2.2)	627.1(±124.6)	621.4(±67.2)	2046.9(±9.7)
	OURS	498.0(±533.5)	588.3(±91.0)	939.3(±170.2)	1096.2(±132.8)	2200.6(±11.4)
	OURS + PPO	481.0(±93.4)	318.9(±19.8)	935.0(±543.6)	839.5(±31.0)	2112.5(±18.7)
Pure-ϵ-40	Persim	1992.5(±1008.7)	296.0(±25.3)	768.5(±289.7)	717.6(±40.2)	214.3(±15.4)
	Persim-M	514.0(±41.5)	198.7(±29.6)	2313.7(±904.4)	2313.0(±52.2)	93.2(±13.2)
	Vanilla CaDM	1863.0(±1181.8)	296.6(±21.7)	-120.3(±42.6)	75.8(±11.7)	-312.0(±39.9)
	PE-TS CaDM	1739.4(±3433.7)	193.5(±12.8)	733.7(±452.9)	553.7(±19.7)	-51.8(±44.3)
	OURS	2224.3(±481.9)	395.5(±27.8)	916.7(±522.3)	730.5(±46.3)	485.8(±48.0)
	OURS + PPO	2183.7(±1889.2)	328.8(±21.1)	1076.6(±1679.7)	758.3(±24.8)	470.7(±55.8)
Random	Persim	400.3(±131.4)	400.6(±275.3)	1262.9(±670.8)	2601.8(±18.9)	2329.9(±15.7)
	Persim-M	229.5(±115.3)	1390.7(±1319.0)	2601.5(±44.8)	-55.2(±20.8)	101.7(±135.2)
	Vanilla CaDM	97.9(±81.9)	67.3(±447.1)	537.1(±2669.2)	2535.7(±17.5)	2061.3(±30.1)
	PE-TS CaDM	157.7(±56.2)	-95.7(±2025.1)	1088.8(±1377.0)	2145.6(±3.0)	1783.8(±1.2)
	OURS	878.5(±269.8)	1705.7(±953.4)	3294.7(±705.1)	2772.0(±15.7)	2723.1(±3.3)
	OURS + PPO	545.7(±376.6)	680.9(±56.8)	1462.3(±161.1)	2598.7(±29.4)	2520.5(±6.1)

Table 19: Average Reward: HalfCheetah(Mass Scale)

Data	Method	Dynamic1(0.3)	Dynamic2(0.6)	Dynamic3(1.0)	Dynamic4(1.4)	Dynamic5(1.7)
Pure	Persim	62.5(±104.2)	106.6(±40.2)	-149.5(±9.4)	-44.8(±6.5)	-114.1(±13.7)
	Persim-M	-47.9(±11.4)	-92.2(±24.7)	-161.2(±2.0)	-132.1(±2.2)	-110.4(±2.5)
	Vanilla CaDM	-19.6(±61.2)	-55.3(±37.0)	-1026.8(±79.3)	-68.6(±6.8)	-373.0(±8.1)
	PE-TS CaDM	-251.0(±29.2)	16.8(±30.3)	-154.5(±7.2)	-346.2(±2.7)	-204.1(±19.2)
	OURS	264.7(±33.6)	280.5(±117.3)	59.7(±4.4)	22.3(±11.4)	-1.8(±18.7)
	OURS + PPO	189.7(±193.6)	178.2(±48.3)	63.9(±1.9)	76.8(±4.8)	3.6(±5.5)
Pure-ϵ-20	Persim	296.6(±230.6)	272.4(±18.7)	732.3(±243.9)	737.6(±30.2)	2689.7(±19.8)
	Persim-M	397.8(±33.0)	-37.0(±13.2)	541.4(±14.1)	466.2(±19.9)	336.9(±14.1)
	Vanilla CaDM	201.1(±231.7)	69.4(±42.3)	527.1(±466.0)	452.2(±74.7)	2037.7(±9.0)
	PE-TS CaDM	198.0(±226.6)	-257.4(±3.3)	641.6(±143.2)	610.6(±80.2)	2336.3(±9.8)
	OURS	570.2(±592.2)	638.1(±95.6)	809.0(±121.9)	1057.7(±109.9)	2431.0(±9.8)
	OURS + PPO	541.9(±91.5)	252.5(±15.8)	977.0(±462.9)	877.8(±27.9)	1809.9(±16.0)
Pure-ϵ-40	Persim	1939.8(±927.2)	315.1(±28.6)	743.1(±249.1)	631.1(±30.9)	266.8(±22.1)
	Persim-M	634.2(±60.6)	194.8(±24.3)	2261.2(±957.2)	2507.2(±57.7)	103.9(±12.3)
	Vanilla CaDM	1770.4(±1447.9)	306.9(±21.5)	-130.3(±54.1)	83.2(±12.8)	-255.3(±46.9)
	PE-TS CaDM	1920.0(±2857.9)	176.3(±12.6)	721.1(±383.0)	541.6(±22.6)	-50.6(±55.7)
	OURS	2246.9(±529.7)	387.9(±27.9)	808.1(±437.8)	721.5(±33.7)	383.2(±37.2)
	OURS + PPO	2432.4(±2458.1)	353.1(±23.2)	1098.9(±1398.2)	815.2(±26.7)	457.5(±64.8)
Random	Persim	373.6(±153.7)	440.9(±230.0)	1203.9(±525.9)	2444.5(±18.5)	2200.5(±16.3)
	Persim-M	238.4(±140.9)	1223.4(±1327.9)	2918.0(±49.8)	-58.5(±29.8)	107.1(±173.0)
	Vanilla CaDM	97.7(±89.4)	64.0(±552.0)	518.8(±2541.2)	2215.2(±15.5)	2172.3(±33.1)
	PE-TS CaDM	81.1(±71.2)	76.6(±647.5)	430.0(±2238.0)	2180.5(±16.1)	1729.4(±20.9)
	OURS	981.4(±266.7)	1595.1(±683.9)	3722.0(±761.5)	2977.8(±21.7)	2687.4(±2.6)
	OURS + PPO	545.5(±325.9)	687.4(±66.0)	1491.6(±201.9)	2644.5(±22.9)	2538.6(±5.7)

Table 20: Average Reward: HalfCheetah(Damping Scale)

Data	Method	Dynamic1(0.6)	Dynamic2(0.9)	Dynamic3(1.0)	Dynamic4(1.1)	Dynamic5(1.4)
Pure	Persim	182.9(±55.7)	306.4(±19.3)	565.8(±16.6)	431.8(±59.8)	138.8(±71.3)
	Persim-M	51.1(±87.3)	175.9(±18.7)	488.5(±15.3)	341.6(±11.7)	64.6(±77.9)
	Vanilla CaDM	152.6(±26.1)	251.4(±44.3)	455.8(±25.2)	252.3(±68.4)	122.4(±55.6)
	PE-TS CaDM	203.0(±32.2)	347.9(±25.8)	623.4(±22.4)	294.8(±32.9)	169.5(±69.8)
	OURS	272.5(±31.3)	355.3(±10.7)	665.5(±32.4)	442.7(±119.0)	274.3(±81.9)
	OURS + PPO	266.9(±31.9)	327.6(±13.6)	530.7(±41.8)	405.6(±76.1)	267.1(±84.4)
Pure-ϵ-20	Persim	270.6(±54.3)	1549.0(±38.9)	2023.5(±25.2)	1245.1(±131.8)	805.2(±29.8)
	Persim-M	-509.8(±72.1)	465.8(±25.2)	1943.5(±15.2)	1309.3(±22.9)	795.2(±12.5)
	Vanilla CaDM	-582.4(±39.8)	252.2(±27.0)	1567.3(±15.5)	349.3(±13.9)	51.4(±37.5)
	PE-TS CaDM	478.2(±91.8)	834.5(±25.0)	2099.0(±24.6)	1412.0(±44.2)	307.2(±49.4)
	OURS	575.4(±76.0)	1717.1(±62.9)	2975.6(±71.8)	1433.3(±106.1)	1003.7(±87.5)
	OURS + PPO	572.8(±102.7)	1670.4(±75.6)	2174.3(±49.6)	1296.2(±75.0)	974.7(±63.7)
Pure-ϵ-40	Persim	1069.9(±32.1)	2357.0(±22.3)	3086.0(±13.9)	2182.8(±18.9)	1264.4(±50.9)
	Persim-M	305.8(±45.8)	2018.3(±28.4)	2110.9(±34.5)	2364.0(±28.1)	178.9(±83.9)
	Vanilla CaDM	-501.6(±78.8)	2215.9(±18.2)	1915.2(±26.0)	1885.3(±44.3)	-292.7(±75.1)
	PE-TS CaDM	930.2(±41.1)	1845.1(±29.2)	2826.7(±17.0)	2370.5(±22.9)	1021.4(±68.1)
	OURS	1175.4(±168.4)	2707.0(±30.6)	3830.6(±19.2)	2485.5(±33.1)	1459.8(±42.6)
	OURS + PPO	1137.5(±156.3)	2318.8(±27.0)	3327.5(±19.7)	2309.0(±39.1)	1327.4(±59.2)
Random	Persim	473.1(±23.8)	1782.2(±36.7)	1893.3(±15.2)	1987.3(±32.0)	574.9(±65.1)
	Persim-M	-614.2(±42.5)	1208.4(±16.7)	1124.9(±8.1)	1220.4(±15.6)	343.7(±37.8)
	Vanilla CaDM	-1322.9(±64.1)	470.4(±21.5)	767.9(±7.0)	736.1(±15.5)	-485.8(±47.7)
	PE-TS CaDM	161.7(±39.0)	1354.5(±16.3)	1402.9(±15.4)	1839.0(±4.5)	598.0(±29.4)
	OURS	505.8(±24.0)	1960.7(±66.1)	2236.4(±14.8)	1987.9(±18.3)	1261.9(±25.8)
	OURS + PPO	298.7(±22.9)	2004.6(±101.6)	2030.9(±12.2)	1894.9(±13.3)	1174.2(±31.3)

Table 21: Average Reward: SlimHumanoid(Mass Scale)

Data	Method	Dynamic1(0.6)	Dynamic2(0.9)	Dynamic3(1.0)	Dynamic4(1.1)	Dynamic5(1.4)
Pure	Persim	199.0(±41.2)	301.7(±17.1)	559.5(±18.5)	402.1(±42.3)	121.6(±71.0)
	Persim-M	53.2(±66.3)	171.9(±25.1)	534.4(±10.8)	388.8(±12.0)	56.0(±74.3)
	Vanilla CaDM	163.7(±34.2)	266.2(±46.0)	412.6(±25.8)	238.6(±80.0)	140.4(±57.4)
	PE-TS CaDM	208.0(±24.8)	388.9(±23.3)	598.9(±24.1)	288.3(±35.0)	166.8(±77.1)
	OURS	292.7(±31.8)	303.3(±11.6)	459.8(±35.6)	489.8(±92.2)	290.6(±75.5)
	OURS + PPO	258.7(±23.4)	297.6(±10.6)	547.0(±32.1)	367.2(±87.1)	273.1(±71.8)
Pure-ϵ-20	Persim	275.1(±56.2)	1707.0(±38.7)	1683.9(±22.6)	1408.7(±131.8)	886.5(±36.7)
	Persim-M	-512.6(±51.7)	501.2(±27.4)	1965.3(±11.5)	1435.4(±27.3)	848.6(±13.7)
	Vanilla CaDM	-599.8(±53.0)	266.7(±27.8)	1772.3(±13.4)	337.2(±18.1)	57.5(±31.2)
	PE-TS CaDM	533.3(±88.1)	876.8(±26.2)	2409.3(±24.6)	1515.2(±39.0)	245.0(±62.0)
	OURS	534.9(±94.9)	1565.6(±62.4)	2569.2(±66.5)	1292.0(±98.2)	1186.7(±69.5)
	OURS + PPO	526.5(±97.0)	1702.3(±73.7)	2009.5(±52.2)	1229.2(±99.8)	1063.6(±68.5)
Pure-ϵ-40	Persim	954.4(±25.0)	2382.7(±23.1)	3075.1(±14.3)	1876.5(±21.3)	1166.5(±50.3)
	Persim-M	356.1(±38.9)	1968.2(±26.0)	1950.2(±33.4)	2628.0(±35.7)	178.9(±62.7)
	Vanilla CaDM	-550.8(±56.2)	2115.3(±19.6)	1895.4(±23.9)	1879.2(±36.4)	-286.2(±58.0)
	PE-TS CaDM	970.3(±29.8)	2054.7(±31.3)	2652.7(±18.0)	2319.6(±28.8)	922.4(±75.7)
	OURS	1062.0(±151.0)	2505.7(±23.9)	3581.9(±21.3)	2453.0(±28.3)	1460.6(±48.1)
	OURS + PPO	1251.9(±136.4)	2417.4(±25.4)	3469.3(±21.1)	2056.8(±39.2)	1135.7(±54.5)
Random	Persim	488.9(±18.9)	1745.4(±32.6)	2051.7(±19.1)	1917.2(±32.0)	533.9(±74.5)
	Persim-M	-526.4(±50.1)	1284.4(±17.5)	1129.0(±7.3)	1225.7(±16.0)	331.6(±33.4)
	Vanilla CaDM	-1355.8(±55.4)	498.6(±23.1)	709.3(±7.6)	732.4(±16.7)	-510.8(±61.7)
	PE-TS CaDM	178.9(±42.8)	1449.8(±14.3)	1514.9(±16.5)	1666.8(±3.5)	581.4(±29.8)
	OURS	282.4(±19.5)	1821.7(±71.7)	2044.7(±10.1)	1659.4(±15.9)	1108.3(±26.1)
	OURS + PPO	291.8(±19.4)	2095.7(±76.3)	2179.2(±10.9)	2040.8(±15.2)	1325.8(±36.2)

Table 22: Average Reward: SlimHumanoid(Damping Scale)