Model-based Meta Reinforcement Learning using Graph Structured Surrogate Models and Amortized Policy Search

Qi Wang [1] Herke van Hoof [1]

Abstract

Reinforcement learning is a promising paradigm for solving sequential decision-making problems, but low data efficiency and weak generalization across tasks are bottlenecks in real-world applications. Model-based meta reinforcement learning addresses these issues by learning dynamics and leveraging knowledge from prior experience. In this paper, we take a closer look at this framework and propose a new posterior sampling based approach that consists of a new model to identify task dynamics together with an amortized policy optimization step. We show that our model, called a graph structured surrogate model (GSSM), achieves competitive dynamics prediction performance with lower model complexity. Moreover, our approach in policy search is able to obtain high returns and allows fast execution by avoiding test-time policy gradient updates.

1. Introduction

Reinforcement learning (RL) has been successfully applied to several complicated tasks and achieved remarkable performance, even surpassing outstanding human players in a variety of domains (Mnih et al., 2015; Silver et al., 2017; Vinyals et al., 2019). By exploration and exploitation, a collection of sequential decision-making problems can be theoretically addressed in this paradigm.

As a cutting-edge research topic, there still remain long standing challenges when putting RL into practice. In particular, these can be viewed from three aspects: i) *data efficiency*, the prevalent branch of RL algorithms as model free reinforcement learning (MFRL) poses great demands on massive interactions with an environment, making it unrealistic to conduct in most real-world applications (Sutton & Barto, 2018; Chua et al., 2018). ii) *robustness to unseen environments*, when an environment of interest drifts in terms of dynamics or reward mechanisms, previously learned skills suffer the risk of poor generalization (Jing et al., 2018; Clavera et al., 2019). And dynamics mismatch easily leads to Sim2Real problems (Peng et al., 2018). iii) *instantaneously planning*, either model predictive control or calibration in previously learned policies consumes additional time in execution phases (Wang & Ba, 2019). And critical issues might arise in real-time decision-making missions with strict time constraints, e.g. autonomous driving.

To address above mentioned concerns, we propose a graph structured surrogate model (GSSM) together with a strategy of amortized policy search. The work is within the framework of Model-based Meta Reinforcement Learning (MBMRL) (Nagabandi et al., 2019; Sæmundsson et al., 2018; Killian et al., 2017; Lee et al., 2020). Our approach makes an attempt to improve dynamics prediction, accelerate policy learning and enable fast adaptation across tasks via latent variables. Importantly, unlike most existing MBMRL methods using time-expensive derivative-free algorithms in model predictive control, our developed amortized policies do not require adaptation time when faced with a new task.

Our primary contributions are two-fold, respectively in designing flexible meta dynamics models and attaining fast adaptation in policy search in MBMRL:

1. **Graph Structured Surrogate Models.** We develop a novel graph structured dynamics model across tasks, which enables effectively encoding of memories and abstracting environments in a latent space. In comparison to related work (Kim et al., 2019), ours is more lightweight but achieves comparable or better performance in forecasting dynamics.

2. **Amortized Meta Model-based Policy Search.** We propose a new strategy for meta model-based policy search that learns latent variable conditioned policies. This enables fast adaptation without additional policy gradient updates and significantly improves policy performance. Interpretations are given from a perspective of posterior sampling (Osband et al., 2013).

[1]Amsterdam Machine Learning Lab, University of Amsterdam, Amsterdam, the Netherlands. Correspondence to: Qi Wang <q.wang3@uva.nl>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

2. Literature Review

As already mentioned, several critical bottlenecks restrict universal applications of RL algorithms. In terms of mastering new skills rapidly, meta learning is an ideal paradigm to achieve with a few instances. As for data efficiency, both model-based reinforcement learning (MBRL) and meta learning can reduce sample complexity.

Meta Learning. The core of meta learning is to discover implicit common structures across a collection of similar tasks and then generalize such knowledge to new scenarios. Leveraging knowledge from a meta learner to a task-specific learner is called *fast adaptation*. Two strategies are commonly used for meta learning, respectively Gradient-based Meta Learning and Contextual Meta Learning. A representative framework for gradient-based meta learning is model agnostic meta learning (MAML) (Finn et al., 2017; Flennerhag et al., 2019; Lee & Choi, 2018), where both a meta learner and an adaptor are derived via gradient information after a few shots in a specific task. The contextual meta learning algorithms rely on task specific latent variables to identify a task after a few observations. This strategy theoretically does not require gradient adaptation in new tasks but constructing task relevant latent variables is decisive (Garnelo et al., 2018a;b; Hausman et al., 2018).

Model-based Reinforcement Learning. Key to applications within a RL framework is how to boost sample efficiency, and MBRL serves a role as approximating a target environment for the agent to interact with. In an environment with unknown dynamics, MBRL either learns a deterministic map or a distribution of transitions $p(\Delta s|[s,a])$. Generally, deterministic modelling on dynamical systems does not involve random variables in the hidden units, and some auto-regressive neural network structures are quite typical in this family (Leibfried et al., 2016; Nagabandi et al., 2017; Amos et al., 2018; van der Pol et al., 2020). Stochastic modelling on dynamical systems are mainly formulated by incorporating uncertainty in system parameters and observation noise (Deisenroth & Rasmussen, 2011; Kamthe & Deisenroth, 2017; Hafner et al., 2018; Chua et al., 2018).

Meta Reinforcement Learning. Most of meta RL algorithms follow a model-free paradigm, e.g. MAESN (Gupta et al., 2018), RL2 (Duan et al., 2016), Learn2Learn (Wang et al., 2016) and PEARL (Rakelly et al., 2019). But experimental results show that these methods work poorly with limited training samples. To obtain satisfying performance with lower sample complexity, researchers are focusing on the combination of meta learning and MBRL. Nagabandi et al. (2019) takes a gradient-based strategy as MAML and alleviates the gap of Sim2Real. In (Sæmundsson et al., 2018), Gaussian process latent variable models perform task inference and learn dynamics across tasks. CaDM (Lee et al., 2020) is a novel SOTA MBMRL method, which includes forward and backward models to more effectively utilize sequential dynamics. Another model strongly related to ours is in (Galashov et al., 2019), where neural processes (NPs) are used to identify dynamics of tasks, but it requires to re-train or fine-tune parameterized policies via gradient updates in new tasks. Note that most of these MBMRL methods focus on *fast adaptation* in *dynamics models*. These either make use of *derivative-free* algorithms for model predictive control or re-train policies in separate tasks, which requires additional computational cost on environments with higher dimensionality (Wang & Ba, 2019). In AdMRL (Lin et al., 2020), task-specific policies are optimized using an implicit function theorem, but it considers the case when dynamics are shared across tasks with different goals. In MIER (Mendonca et al., 2020), labels are updated by querying a neural network of a dynamics system, which is efficient in practice. In our method, we amortize this step and further reduce computations in adaptation

3. Problem Formulation and Preliminaries

The decision-making process in RL is usually characterized with a discrete-time Markov Decision Process (MDP), denoted by $\mathcal{M}$. Given states $s_t \in \mathcal{S}$, actions $a_t \in \mathcal{A}$, policy functions π, state transition distributions $\mathcal{P}$, reward functions $\mathcal{R}$ and a discount factor γ for a step-wise reward, a MDP can be formalized with a tuple of these elements $\mathcal{M}_k = (\mathcal{S}, \mathcal{A}, \mathcal{P}_k, \mathcal{R}_k, \gamma)$. The return of cumulative rewards is a summation of discounted reward feedback $r(s_t, a_t)$ along the trajectories $\tau := (s_0, a_0, r_0, \dots, s_{H-1}, a_{H-1}, r_{H-1}, s_H)$.

The optimization objective in model-free RL methods is to find policies that maximize the expected cumulative rewards over trajectories. In contrast, MBMRL considers a distribution over MDPs $p(\mathcal{M})$, and the goal is to simultaneously build dynamics models and act optimally w.r.t. the learned dynamics models.

3.1. Optimization Objective in MBMRL

More formally, we study MBMRL problems from the optimization perspective and formulate the following two correlated objectives.

$$\max_{\theta} \mathbb{E}_{\substack{\mathcal{M}\sim p(M) \\ ([s,a],s')\sim\mathcal{M}}} \ln\left[p_{\theta_{\mathcal{M}}}(s'|[s,a])\right], \text{ s.t. } p_{\theta_{\mathcal{M}}} = u(\theta, \mathcal{D}^{\text{tr}}_{\mathcal{M}}) \tag{1a}$$

$$\max_{\varphi_{\mathcal{M}}} \mathbb{E}_{\substack{s'\sim p_{\theta_{\mathcal{M}}}(s'|[s,a]) \\ a\sim\pi_{\varphi_{\mathcal{M}}}}} \left[\sum_{t=0}^{H-1} \gamma^t r_{\mathcal{M}}(s_t, a_t)\right], \mathcal{M}\sim p(M) \tag{1b}$$

Here Eq. (1a) is to maximize the log-likelihood of state-transitions $p(s'|[s,a])$ in a collection of MDPs and u represents a fast adaptation mechanism in $\mathcal{M}$ to learn an updated

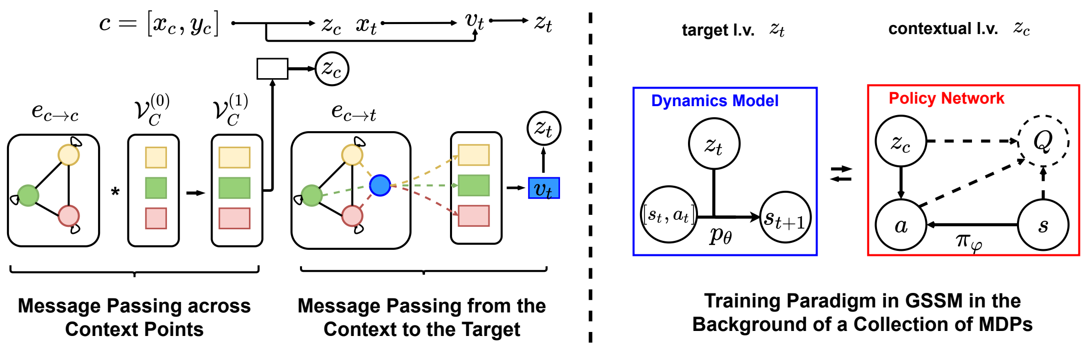


Figure 1. Graph Structured Surrogate Models. On the **Left**: The colored □ denotes a node feature vector in our fully connected graphs with nodes ◯, and lines between ◯ record pairwise similarities `sim`. $\mathcal{V}_C^{(0)}$ denotes the initial feature matrix of all context nodes. Information Flows in **Dynamics Models** from the left to the right describe the Message Passing between context points c (Edges in solid lines $e_{c\to c}$) and to the **target point** x_t (Edges in dashed lines $e_{c\to t}$) to obtain the transformed node feature matrix $\mathcal{V}_C^{(1)}$. On the **Right**: In amortized policy search, latent variables participate in both approximate **Dynamics Models** and **Policy Networks** (Dashed elements are involved in the module when using Actor-Critic frameworks and Double arrows mean interactions to learn amortized policies).

dynamics model $p_{\theta_{\mathcal{M}}}$ with meta-learned parameters θ and a few transition instances $\mathcal{D}^{\text{tr}}_{\mathcal{M}}$. Eq. (1b) corresponds to learning a policy $\pi_{\varphi_{\mathcal{M}}}$ or finding a planning strategy in separate dynamics models. It is worth noting that our objective of MBMRL differs from previous work, and it consists of two phases as *dynamics model learning* and *policy optimization*.

3.2. MBMRL with Latent Variables

Latent variables play diverse roles in meta RL (Gal et al., 2016; Rakelly et al., 2019). Here we focus on a branch of MBMRL methods, which uses latent variables to help formulate meta dynamics models. Mostly, a latent variable z is inferred from a few shots of transitions to summarize statistics of a specific environment $\mathcal{M}$ (Garnelo et al., 2018b;a). Then the learned latent variables participate in dynamics prediction $p_\theta(\Delta s|[s,a], z)$ and approximate dynamics of different MDPs (Galashov et al., 2019; Lee et al., 2020).

The meta learning surrogate model (MLSM) (Galashov et al., 2019) is an example of latent variable MBMRL methods, which is closest to ours in literature. The neural processes (Garnelo et al., 2018b) work as meta dynamics models in MLSM. But we notice that in MLSM: (1) a simple *mean pooling* over context points to obtain latent variables is difficult to utilize the relevance between the context and the target transition samples for all data points' prediction; (2) computationally expensive policy gradient updates are required in policy search when faced with a new task.

4. Graph Structured Surrogate Models with Amortized Policy Search

In this section, we aim to address the shortcomings of insufficiently expressive models and expensive policy gradient optimization by MLSM.

In order to do so, we at first develop a graph structured surrogate model to learn representations of latent variables via message passing, which better approximates local dynamics of MDPs. Then an amortized policy search strategy is introduced to enable fast policy adaptation without gradient updates in new tasks.

In GSSM, two types of latent variables are learned. Fig. (1), which will be explained fully in the following section, illustrates target latent variables z_t to predict individual state transition and a global latent variable z_c to encode the context for policies to condition. For the sake of simplicity, we denote the dynamics model input by $x = [s, a]$ and the dynamics model output by $y = \Delta s$.

4.1. Graph Structured Latent Variables

For the transition dataset from a task, we have a set of context points $[x_c, y_c]$ and the target point $[x_t, y_t]$. Similar to other context-based meta learning algorithms (Garnelo et al., 2018b), $[x_c, y_c]$ are observed state transitions used to infer the task. We treat these context points in a form of graph structured dataset $\mathcal{G} =< \mathcal{V}, \mathcal{E} >$, which comprises of a collection of vertices $\mathcal{V} = [x_c, y_c]$ and relational edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ (notations edge e and vertex value v in Fig. (1)).

Here a fully connected graph is built to characterize the pairwise relationship. The value as the initial node feature vector refers to $[x_i, y_i]$, and the construction of a graph Laplacian matrix is based on the pairwise similarities between samples $\texttt{sim}(x_i, x_j) = \frac{\langle u(x_i), u(x_j)\rangle}{\|u(x_i)\|_2 \cdot \|u(x_j)\|_2}$, where $u(x)$ is the embedding of a sample input x using neural networks, and the notation $\langle \cdot, \cdot \rangle$ means a dot product. All of these are illustrated on the left side of Fig. (1).

Message Passing between Context Points. This process is to deliver and aggregate neighborhood information for node representations, which can be specified via the normalized Laplacian as follows.

$$\hat{s}_{ij} = \frac{\exp\left(\beta \cdot \mathtt{sim}(x_i, x_j)\right)}{\sum_{j \in \mathcal{O}_i} \exp\left(\beta \cdot \mathtt{sim}(x_i, x_j)\right)} \tag{2a}$$

$$h_i^{(l+1)} = \sigma\left(W^{(l)} h_i^{(l)} + \sum_{j \in \mathcal{O}_i} \hat{s}_{ij} W^{(l)} h_j^{(l)}\right) \tag{2b}$$

where β is a tunable parameter for pairwise similarities, $W^{(l)}$ is the network parameter for node feature transformations in l-th layer, $\hat{s}_{ij}$ is the element in a normalized Laplacian matrix L, and $h_i^{(l)}$ is the l-th intermediate node feature vector after message passing from sample i's neighborhoods $\mathcal{O}_i$ to itself.

In Eq. (2), a self-loop is added for message passing. The node j's feature vector after the final message passing are denoted by h_j. With the operation of message passing in Eq. (3), a learned representation for each node summarizes statistics of interactions,

$$v_i = \sum_{j \in \mathcal{O}_i} \hat{s}_{ij} h_j,\ g_c = \bigoplus_{i \in \mathcal{V}} v_i,\ q_{\phi_2}(z_c) = \mathcal{N}\big(\mu(g_c), \Sigma(g_c)\big) \tag{3}$$

where $\bigoplus$ denotes a mean pooling operation over all node feature vectors and the last term describes the amortized distribution (Zhang et al., 2018) for context points.

Message Passing from the Context to the Target. This process is to transmit task-beneficial information from the context to the target. And the representation of the node i is v_i in Eq. (3). Considering relevance to the target point $[x_t, y_t]$ varies from instance to instance, we compute the weight for each context point x_i, denoted by $\hat{s}_{it}$, via Eq. (2.a) to measure the relevance. Then the aggregated context message can be represented as g_t in a weighted way.

$$g_t = \sum_{i \in \mathcal{O}_t} \hat{s}_{it} v_i, \quad q_{\phi_1}(z_t) = \mathcal{N}\big(\mu(g_t), \Sigma(g_t)\big) \tag{4}$$

After the message passing from the context to the target, g_t is further mapped into mean and variance parameters of a proposal distribution $q_{\phi_1}(z_t)$ using neural networks, as displayed on the right side of Eq. (4). Here we employ mean field amortized inference, using diagonal Gaussian distributions for the convenience of computations. Also note that ϕ_1 and ϕ_2 share part of graph neural net parameters, e.g. parameters in feature embeddings, but denote separate variational parameters.

4.2. Approximate Inference & Scalable Training in GSSM

To learn system dynamics with latent variables, we need to specify an objective in optimization together with a predictive distribution $p(y_t|x_t, x_c, y_c)$. Here we sample the context data points $[x_c, y_c]$ together with the target $[x_t, y_t]$ from meta learning dataset $p(D)$.

Though the exact inference for this predictive distribution is intractable, a plausible way is to use the above mentioned variational distribution $q_{\phi_1}(z_t|x_t, x_c, y_c)$ in Eq. (4). As a result, the evidence lower bound (ELBO) is formulated on the right side of Eq. (5).

$$\begin{aligned}\mathbb{E}_{p(D)}\left[\ln p(y_t|x_t, x_c, y_c)\right] \geq \mathbb{E}_{p(D)}\left[\mathbb{E}_{q_{\phi_1}}[\ln p_\theta(y_t|x_t, z_t)]\right] \\ -\mathbb{E}_{p(D)}\left[D_{KL}[q_{\phi_1}(z_t|x_t, x_c, y_c) \parallel p(z_t)]\right]\end{aligned} \tag{5}$$

Similar to (Denton & Fergus, 2018; Pertsch et al., 2020; Garnelo et al., 2018b), a variational distribution $q_{\phi_2}(z_c|x_c, y_c)$ is selected as a prior distribution $p(z_t)$ in ELBO to specify a dynamical system from context points and further *used in the following amortized policy search*. As a result, the induced objective with a learned prior distribution is as follows.

$$\begin{aligned}\mathbb{E}_{p(D)}\left[\ln p(y_t|x_t, x_c, y_c)\right] \geq \mathbb{E}_{p(D)}\left[\mathbb{E}_{q_{\phi_1}}[\ln p_\theta(y_t|x_t, z_t)]\right. \\ \left.-D_{KL}[q_{\phi_1}(z_t|x_t, x_c, y_c) \parallel q_{\phi_2}(z_c|x_c, y_c)]\right]\end{aligned} \tag{6}$$

In implementations, the Monte Carlo estimation is performed for the negative ELBO to obtain Eq. (7),

$$\begin{aligned}\mathcal{L}(\theta, \phi_1, \phi_2) = -\frac{1}{K}\sum_{b=1}^{B}\sum_{k=1}^{K} \ln p_\theta(y_t^{(b)}|x_t^{(b)}, z_t^{(b,k)}) \\ +D_{KL}\left[q_{\phi_1}(z_t^{(b)}|x_t^{(b)}, x_c^{(b)}, y_c^{(b)}) \parallel q_{\phi_2}(z_c^{(b)}|x_c^{(b)}, y_c^{(b)})\right]\end{aligned} \tag{7}$$

where B is the batch size of samples in meta training, K is the number of particles in estimation, and latent variable values are sampled from the approximate posterior $z_t^{(b,k)} \sim q_{\phi_1}(z_t^{(b)}|x_t^{(b)}, x_c^{(b)}, y_c^{(b)})$.

Similarly, when it comes to prediction using the learned dynamics model, the Monte Carlo estimator can be applied again to derive a predictive distribution in Eq. (8) with the approximate posterior q_{ϕ_1} and collected context points $[x_c, y_c]$.

$$\begin{aligned}p(y_t|x_t, x_c, y_c) = \int q_{\phi_1}(z_t|x_t, x_c, y_c) p_\theta(y_t|x_t, z_t) dz_t \\ \approx \frac{1}{K}\sum_{k=1}^{K} p_\theta(y_t^{(k)}|x_t, z_t^{(k)})\end{aligned} \tag{8}$$

4.3. Amortized Policy Search

Once a dynamics model is learned, policy search can be executed by interacting with a learned dynamics model. Here we concentrate on *fast adaptation of policies* in MBMRL and introduce the concept of *amortized policy search*. This is different from previous planning or policy optimization objectives in separate dynamics models.

To this end, we utilize the strategy of posterior sampling (Osband et al., 2013; Rakelly et al., 2019) to capture task-specific policies. Specifically, a collection of approximate MDPs are sampled from the posterior of task-specific dynamics models, and the agent acts optimally w.r.t. these models. Finding optimal policies for different tasks are time consuming in previous approaches, which require either *re-training* or *fine-tuning meta-learned policies* $\pi_\varphi(a|s)$ in separate dynamics models (Galashov et al., 2019). So we propose to induce task-specific policies $\pi_\varphi(a|[s,z_c])$ by sampling z_c from task relevant latent variable distributions $q_{\phi_2}(z_c|x_c,y_c)$, and optimize policies w.r.t. sampled approximate MDPs. The process of finding these task-specific optimal policies is termed as *amortized policy search* and the obtained policy $\pi_\varphi(a|[s,z_c])$ after meta training is called an *amortized policy*.

In our settings, meta model-based policy search is considered in a distribution of learned approximate dynamics models $p(\hat{\mathcal{M}};\theta,\phi)$. A parameterized policy π_φ is used to collect trajectories τ from a learned dynamics model and evaluate rewards of policies in Eq. (9) to maximize.

$$\mathcal{J}(\varphi;\theta,\phi)=\iint p(\hat{\mathcal{M}};\theta,\phi)p(\tau|\hat{\mathcal{M}};\varphi,\phi)\mathcal{R}(\tau)d\tau d\hat{\mathcal{M}} \tag{9}$$

The following two model-based policy search strategies are commonly-used in this domain, so we modify these to enable the use of amortized policies in MBMRL. Importantly, we take more interest in computing gradients of policies w.r.t. policy parameters φ.

(1) Direct policy search via BPTT. The objective of back-propagation through time (BPTT) (Deisenroth & Rasmussen, 2011; Parmas et al., 2018) in MBMRL can be written in the form of Monte Carlo estimation as follows.

$$\nabla_\varphi\mathcal{J}(\varphi)\approx\frac{1}{BK}\sum_{b=1}^{B}\sum_{k=1}^{K}\left(\nabla_\varphi\sum_{t=0}^{T-1}\gamma^t r_{t+1}^{(b,k)}\right) \tag{10}$$

where $\tau=[s_0,a_0,s_1,r_1,\dots]$, $a_t\sim\pi_\varphi(\cdot|[s_t,z_c])$, $s_{t+1}\sim p_\theta(s_{t+1}|[s_t,a_t],z_t)$, B is the number of batch in tasks, and K is the number of sampled simulated trajectories for each task.

(2) Actor-Critic policy gradient optimization. The induced policy gradient w.r.t Eq. (9) is estimated as Eq. (11) under our amortized policy search method, where A_t is the computed advantage function.

$$\nabla_\varphi\mathcal{J}(\varphi)=\mathbb{E}_{\substack{\hat{\mathcal{M}}\sim p(\hat{\mathcal{M}};\theta,\phi)\\ \tau\sim p(\tau|\hat{\mathcal{M}};\varphi,\phi)}}\Big[\sum_{t=0}^{T-1}\nabla_\varphi\ln\pi(a_t|[s_t,z_c]) \cdot A_t([s_t,z_c],a_t)\Big] \tag{11}$$

Besides, the optimization of value function approximation is inside policy search during meta-training processes via gradient updates. For the sake of simplicity, we skip this step in Algorithm (1) and the optimization is executed in `step` (10) as default. For more details about Actor-Critic settings, please refer to Appendix (F) for more details.

Note that the universal value function approximator augments MDPs with goal information to identify diverse tasks (Schaul et al., 2015). Similarly, in our settings, the task is inferred from context points, and the transition sample is augmented with the inferred task/goal as $\{([s_t,z_c],a_t,r_t)\}_{t=1}^{T}$. The approximate value function $Q_{\pi_\varphi}([s,z_c],a)$ or $V_{\pi_\varphi}([s,z_c])$ is task-specific. In this case, our proposed amortized policy search can be interpreted as finding a universal value function approximator in MDPs of various dynamics.

Algorithm 1 Meta-Training Process.

```
Input  : MDP distribution $\rho(\mathcal{M})$; Batch of tasks $\mathcal{B}$; Exploration policy $\pi_e$.
Output : Trained parameters $\phi$, $\theta$ and $\varphi$.
Initialize parameters $\phi$, $\theta$ and $\varphi$
while Meta-Training not Completed do
    // meta train dynamics models
    Sample a batch of tasks $\{\mathcal{M}_k\}_{k=1}^{\mathcal{B}}$ from a distribution $\rho(\mathcal{M})$
    Perform roll-outs to collect transition dataset $\mathcal{D}^{\mathcal{B}}=\{\mathcal{D}_k\}_{k=1}^{\mathcal{B}}$ with $\pi_e$
    Optimize $\{\phi,\theta\}$ on $\mathcal{D}^{\mathcal{B}}$ in Eq. (8) to obtain $\{\hat{\mathcal{M}}_k\}_{k=1}^{\mathcal{B}}$
    // meta model-based policy search
    for $i=1,2,\dots,N$ do
        Sample initial states from $\{\hat{\mathcal{M}}_k\}_{k=1}^{\mathcal{B}}$
        Collect episodes by interacting with $\{\hat{\mathcal{M}}_k\}_{k=1}^{\mathcal{B}}$ using $\pi_\varphi$ and update dynamics buffers $\mathcal{D}^{\mathcal{B}}$
        Evaluate rewards in Eq. (9)
        Optimize the policy $\pi_\varphi$: $\varphi\leftarrow\varphi+\alpha\nabla_\varphi\mathcal{J}$.
    end
end
```

5. Experiments and Analysis

To evaluate our approach, we perform experiments in environments with diverse task-specific dynamics. Meanwhile

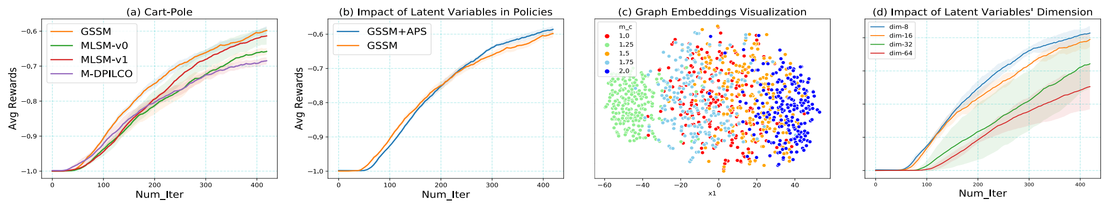


Figure 2. Experimental Results on Cart-Pole Tasks. (a) Learning curves of MBMRL algorithms using non-latent variable conditioned policies. (b) Learning curves of GSSM using (non-)latent variable conditioned policies. (c) t-SNE visualizations of latent variables in GSSM (samples are from 5 tasks with a pole mass as 1.0 and cart masses as $\{1.0, 1.25, 1.5, 1.75, 2.0\}$). (d) Learning curves of GSSM+APS with different dimensions of latent variables. For all learning curves, the averaged rewards are tested after each `iter` in an offline way and results indicate means and a standard error of the mean in 5 runs.

Algorithm 2 Meta-Testing Phases.

```
Input  : Meta-trained $\phi$, $\theta$ and $\varphi$ ; Null buffer $\mathcal{B}$; Adaptation steps $K$.
Output : Cumulative rewards of episodes.
Sample a testing task $\mathcal{M}_*\sim\rho(\mathcal{M})$
if Use GSSM and Amortized Policy Search then
    Run $\pi_e$ to collect the memory $\tau : \mathcal{B} \leftarrow \mathcal{B} \cup \{\tau\}$
    Evaluate $\pi_\varphi(a|s, z)$ with $z \sim q(z_c|\mathcal{B})$ in Eq. (9) in $\mathcal{M}_*$.
else
    Run $\pi_e$ to collect the memory $\tau : \mathcal{B} \leftarrow \mathcal{B} \cup \{\tau\}$
    for $i = 1, 2, \ldots, K$ do
        Sample an initial state from the learned $\hat{\mathcal{M}}_*$
        Collect an episode in $\hat{\mathcal{M}}_*$ using $\pi_\varphi(a|s)$
        Evaluate return in Eq. (9)
        Optimize the policy $\pi_\varphi$: $\varphi \leftarrow \varphi + \alpha\nabla_\varphi\mathcal{J}$.
    end
    Evaluate fine-tuned $\pi_\varphi(a|s)$ in $\mathcal{M}_*$.
end
```

the implementation of our developed approach is available in Appendix (G).

5.1. General Settings

In all MBMRL related experiments, meta training and testing phases respectively follow that in Algorithm (1)/(2), where $\hat{\mathcal{M}}$ is an approximate dynamics model. Similar to that in (Galashov et al., 2019), the exploratory policy π_e is completely random without parameters to initialize the transition buffer or collect transitions for identifying the task in meta testing phases.

Some baseline methods include:

- **L2A** (Nagabandi et al., 2019). As a gradient-based meta RL approach, the Learning to Adapt (L2A) utilizes a MAML paradigm to learn dynamics and adaptation strategies.
- **MLSM-v0** (Galashov et al., 2019). Meta Learning Surrogate Model (MLSM) makes use of neural processes in MBMRL, where latent variables are incorporated to identify tasks.
- **MLSM-v1** (Galashov et al., 2019; Kim et al., 2019). This is a boosted version of MLSM-v0, where an attention neural network is added to learn sample dependent latent variables.
- **M-DPILCO** (Gal et al., 2016). Deep PILCO, which uses Bayesian neural networks (BNNs) to fit dynamics. Ensemble of episodes from BNNs are used for policy optimization.

For GSSM/M-DPILCO/MLSM-v0/MLSM-v1, we use the same policy search strategy: a policy $\pi_\varphi(a|s)$ is optimized across a collection of approximate dynamics models in Algorithm (1) and in testing processes this policy is fine-tuned via policy gradient updates as fast adaptation in separate dynamics models like that in Algorithm (2). For GSSM+APS, we use a latent variable conditioned policy $\pi_\varphi(a|s, z_c)$ to enable amortized policy search (APS) in GSSM meta dynamics models. As for L2A, the implementation follows that in (Nagabandi et al., 2019), each learned dynamics model after fast adaptation via gradient updates is used to plan separately.

With these models, we use formerly introduced meta model-based policy search strategies to combine: (i) direct policy search trained via BPTT (Parmas et al., 2018) (only applied to Cart-Pole environments) (ii) actor-critic policy search using proximal policy optimization (PPO) (Schulman et al., 2017) (applied to all the other environments).

Meanwhile, **DR-PPO** is included as a model-free RL baseline, where PPO is trained across tasks via Domain Randomization. Another algorithm as the probabilistic embedding for actor-critic RL (referred to as **PE-PPO**) is also introduced in comparisons, which follows the same implementation in PEARL algorithms (Rakelly et al., 2019). More details about task settings and available PyTorch modules are given in Appendix (G).

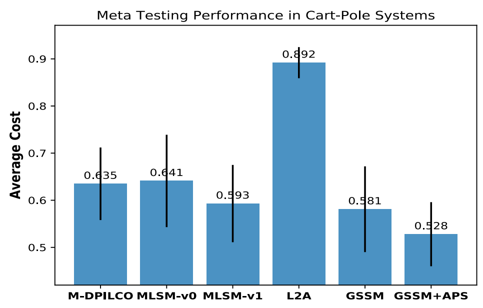


Figure 3. Meta Testing Performance. The average cost is negative of average rewards, and standard deviations are attached as error bars. For methods except GSSM+APS, *additional time* is required to perform gradient-based adaptation in policies.

5.2. Cart-Pole Systems

At first, we evaluate MBMRL models on a Cart-Pole Swing-Up task. The physics system can be found in (Gal et al., 2016; Galashov et al., 2019), and meta tasks are generated in the following way. We respectively sample masses of a cart m_c and a pole m_p from uniform distributions $\mathcal{U}[1.0, 2.0]$ and $\mathcal{U}[0.7, 1.0]$. The mission is to perform actions to reach the goal with the end of the pole. The state is $[x_c, \theta, x'_c, \theta']$, while the action corresponds to the force in a continuous interval $a \sim [-10, +10]\ N$. The horizon in episodes is set to be 25 the same with that in former works.

In **meta training** processes, various tasks are sampled during iterations and results are averaged step-wise rewards. In Fig. (2.(a)), with the same policy search strategies (non-amortized policies), we observe GSSM shows best performance with lower variances in learning curves, followed by MLSM-v1 with larger variances. Also note that GSSM's model complexity is more lightweight than AttnNP in MLSM-v1 while retaining satisfying dynamics prediction capability (Refer to Table (4)/Table (5)). In Fig. (2.(b)), when amortized policies are combined with GSSM, the performance is further advanced. In Fig. (2.(c)), we sample latent values by encoding trajectories from 5 tasks to visualize using t-SNE (Van der Maaten & Hinton, 2008). It can be seen the latent embeddings of different tasks formulate several clusters, which reveal cart masses' impact from latent variables. In Fig. (2.(d)), we also vary the dimension of latent variables and find that when amortized policies are used, a lower dimension results in more compact task embeddings and obtains better results with lower variance.

Note that the principal goal of MBMRL is to generalize previously learned skills to unseen tasks, so we stress the importance of the performance in **meta testing** processes, which is more appropriate to assess the generalization capability. Here 50 unseen tasks are sampled to validate the performance (each task with 50 episodes) and averaged results are displayed in Fig. (3). We observe GSSM and MLSM-v1 are comparable in policy performance but the former has less parameters. As for M-DPILCO and MLSM-v0, they show intermediate performance in testing results. With the same type of dynamics models, GSSM+APS does not require adaptation time and reduces **10%** step-wise costs than GSSM. This suggests amortized policies reveal task relevant information from MDP embeddings and then guide the agent to explore better in separate environments.

Table 1. Meta-testing Results using Model-free Baselines. `MUL` records multiple of required samples used in MBMRL.

ENV	MUL	DR-PPO	PE-PPO
ACROBOT	1X	-0.836(±0.047)	**-0.828(±0.052)**
	3X	-0.433(±0.043)	**-0.420(±0.05)**
H-CHEETAH	1X	**453.4(±150)**	-44.1(±51)
	25X	**1360.5(±130)**	608.2(±73)
S-HUMANOID	1X	**538.5(±91)**	252.4(±260)
	25X	**3533.3(±110)**	1248.1(±150)

5.3. Other Simulation Systems

Other explorations are performed in more complicated simulation systems. These include Acrobot, Half-Cheetah and Slim-Humanoid. For Acrobot, masses m of two pendulums are respectively drawn from uniform distributions as $\mathcal{U}[0.8, 1.2]$ and $\mathcal{U}[0.8, 1.2]$ to configure different tasks with 200 steps as the default horizon (Killian et al., 2017; Sutton & Barto, 2018). For H(alf)-Cheetah/S(lim)-Humanoid from Mujoco (Todorov et al., 2012), we vary mass scales and damping coefficients for different tasks, where the default horizon is 1000 steps.

Main Results in Meta-training Processes. It can be seen in Fig. (4) that with the equal volume of samples, MBMRL models mostly outperform MFRL baselines. Without the use of amortized policies, GSSM shows advantage over other MBMRL baselines. For MLSM-v0/MLSMv-1, intermediate rewards are obtained in Acrobot/H-Cheetah. We also observe unstable results using other baselines in S-Humanoid. When amortized policies are employed, GSSM+APS shows significant performance improvement over all baselines including GSSM. This reflects task-specific information is beneficial in policy optimization. Especially, GSSM+APS reaches around -0.57 equivalent to model-free performance with 3x less time steps in Acrobot. Similar phenomenon can be seen in H-Cheetah, and GSSM+APS mostly starts with poor initialization but gradually surpasses others as latent variables are becoming more and more meaningful.

Main Results in Meta-testing Processes. As exhibited in Table (4), we evaluate predictive performance of dynam-

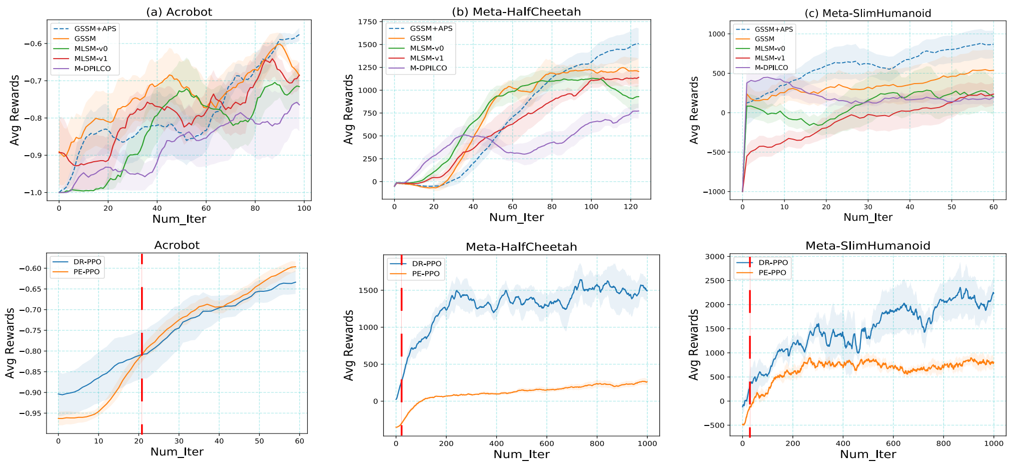


Figure 4. Performance of Policies in Meta-Training Processes. Figures in the first row are MBMRL results while those in the second row are MFRL results (Red vertical dotted lines indicate the threshold of required time-steps used to train MBMRL baselines.). Here environments are varied in terms of dynamics during iterations. The average rewards are tested after each `iter` in an offline way and results indicate means and corresponding standard errors of the mean in 5 runs.

Table 2. Average Rewards in Meta-testing Tasks using Learned Policy Networks. (For each testing task, 50 episodes are sampled and averaged in rewards. Figures in brackets are standard deviations across testing tasks, with bold ones the best.)

ENV	GSSM+APS	GSSM	M-DPILCO	MLSM-V0	MLSM-V1	L2A
ACROBOT	**-0.478(±0.049)**	-0.506(±0.068)	-0.645(±0.06)	-0.560(±0.064)	-0.524(±0.052)	-0.7775(±0.054)
H-CHEETAH	**1597.4(±200)**	1306.6(±140)	862.0(±280)	827.3(±190)	1226.8(±64)	-17.9(±130)
S-HUMANOID	**1641.8(±170)**	717.1(±130)	596.0(±340)	285.9(±360)	745.6(±150)	124.9(±570)

ics models in unseen tasks. After executing paired-*t* test, we find GSSM+APS significantly surpasses MLSM-v0 in Cart-Pole/Acrobot/S-Humanoid and achieves comparable performance to MLSM-v1 in forecasting dynamics. But GSSM is simpler in terms of model complexities.

The corresponding policy performance is also displayed in Table (2). We also notice that with the same policy search strategy, GSSM outperforms other baselines in Acrobot/H-Cheetah. When amortized policies are used, GSSM+APS is significantly superior to MLSM-v1[1]. These findings reflect strong generalization capability in unseen tasks when combining GSSM and amortized policies. L2A works not so well in our environments even after trying several hyperparameters, similar to observations in the work (Hiraoka et al., 2020; Lee et al., 2020), and lower rewards could be due to unstable adaptation in dynamics models. Results in model-free cases are also illustrated in Table (1). PE-PPO performs worse than DR-PPO in two environments, even though probabilistic embeddings of tasks join the policy learning. Here we use permutation invariant amortized distributions to formulate $\pi(a|s,z)$ as that in PEARL (Rakelly et al., 2019), but PEARL algorithms seem sensitive to neural architectures of latent variables. It turns out latent variables in amortized policies can lead to task-specific optimal values (Schaul et al., 2015) in actor-critic policy search but appropriate embeddings are decisive as well.

[1]A paired-t test between GSSM and MLSM-v1 over all tasks is executed, and results are significant with a default significance level 0.05.

6. Discussion and Conclusion

We have proposed a novel meta dynamics model (GSSM), which consists of a local latent variable z_t for individual dynamics prediction and a global latent variable z_c to summarize a MDP for the policy to condition. Learning the representations of these latent variables is achieved via the message passing. GSSM demonstrates its effectiveness in capturing task-specific system dynamics and exhibits good generalization capability across tasks.

Meanwhile, amortized policies are developed for more efficient meta model-based policy search. These policies allow for fast adaptation to meta testing tasks without additional policy gradient updates and show competitive performance. It is important to note that this trait helps us avoid either re-planning or adaptation in policies. So our proposed amortized policy search is more suitable to apply in time-sensitive decision-making missions.

Acknowledgements

We thank ICML anonymous reviewers and meta reviewers for their deep insights and constructive suggestions. We thank Matthew Macfarlane and Niklas Höpner for helpful feedback. Importantly, special thanks to those who support me in life during the tough pandemic period.

References

Amos, B., Dinh, L., Cabi, S., Rothörl, T., Colmenarejo, S. G., Muldal, A., Erez, T., Tassa, Y., de Freitas, N., and Denil, M. Learning awareness models. *arXiv preprint arXiv:1804.06318*, 2018.

Asmuth, J., Li, L., Littman, M. L., Nouri, A., and Wingate, D. A bayesian sampling approach to exploration in reinforcement learning. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pp. 19–26, 2009.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pp. 4754–4765, 2018.

Clavera, I., Nagabandi, A., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C. Learning to adapt: Meta-learning for model-based control. *arXiv preprint arXiv:1803.11347*, 3, 2019.

Deisenroth, M. and Rasmussen, C. E. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472. Citeseer, 2011.

Denton, E. and Fergus, R. Stochastic video generation with a learned prior. In *International Conference on Machine Learning*, pp. 1174–1183, 2018.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. Rl2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org, 2017.

Flennerhag, S., Rusu, A. A., Pascanu, R., Yin, H., and Hadsell, R. Meta-learning with warped gradient descent. *arXiv preprint arXiv:1909.00025*, 2019.

Gal, Y., McAllister, R., and Rasmussen, C. E. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4, pp. 34, 2016.

Galashov, A., Schwarz, J., Kim, H., Garnelo, M., Saxton, D., Kohli, P., Eslami, S., and Teh, Y. W. Meta-learning surrogate models for sequential decision making. *arXiv preprint arXiv:1903.11907*, 2019.

Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. Conditional neural processes. In *International Conference on Machine Learning*, pp. 1704–1713, 2018a.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Whye Teh, Y. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b.

Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., and Levine, S. Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems*, pp. 5302–5311, 2018.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

Hiraoka, T., Imagawa, T., Tangkaratt, V., Osa, T., Onishi, T., and Tsuruoka, Y. Meta-model-based meta-policy optimization. *arXiv preprint arXiv:2006.02608*, 2020.

Jing, M., Ma, X., Sun, F., and Liu, H. Learning and inferring movement with deep generative model. *arXiv preprint arXiv:1805.07252*, 2018.

Kamthe, S. and Deisenroth, M. P. Data-efficient reinforcement learning with probabilistic model predictive control. *arXiv preprint arXiv:1706.06491*, 2017.

Killian, T. W., Daulton, S., Konidaris, G., and Doshi-Velez, F. Robust and efficient transfer learning with hidden parameter markov decision processes. In *Advances in neural information processing systems*, pp. 6250–6261, 2017.

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. Attentive neural processes. *arXiv preprint arXiv:1901.05761*, 2019.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Lee, K., Seo, Y., Lee, S., Lee, H., and Shin, J. Context-aware dynamics model for generalization in model-based reinforcement learning. *arXiv preprint arXiv:2005.06800*, 2020.

Lee, Y. and Choi, S. Gradient-based meta-learning with learned layerwise metric and subspace. In *International Conference on Machine Learning*, pp. 2927–2936, 2018.

Leibfried, F., Kushman, N., and Hofmann, K. A deep learning approach for joint video frame and reward prediction in atari games. *arXiv preprint arXiv:1611.07078*, 2016.

Li, J., Vuong, Q., Liu, S., Liu, M., Ciosek, K., Ross, K., Christensen, H. I., and Su, H. Multi-task batch reinforcement learning with metric learning. *arXiv preprint arXiv:1909.11373*, 2019.

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

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In *ICLR (Poster)*, 2016.

Lin, Z., Thomas, G., Yang, G., and Ma, T. Model-based adversarial meta-reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Mendonca, R., Geng, X., Finn, C., and Levine, S. Meta-reinforcement learning robust to distributional shift via model identification and experience relabeling. *arXiv preprint arXiv:2006.07178*, 2020.

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. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Nagabandi, A., Yang, G., Asmar, T., Kahn, G., Levine, S., and Fearing, R. S. Neural network dynamics models for control of under-actuated legged millirobots. *arXiv preprint arXiv:1711.05253*, 2017.

Nagabandi, A., Clavera, I., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *ICLR (Poster)*, 2019.

Osband, I., Russo, D., and Van Roy, B. (more) efficient reinforcement learning via posterior sampling. In *NIPS*, 2013.

Parmas, P., Rasmussen, C. E., Peters, J., and Doya, K. Pipps: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pp. 4065–4074. PMLR, 2018.

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 1–8. IEEE, 2018.

Pertsch, K., Lee, Y., and Lim, J. J. Accelerating reinforcement learning with learned skill priors. *arXiv preprint arXiv:2010.11944*, 2020.

Rakelly, K., Zhou, A., Finn, C., Levine, S., and Quillen, D. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pp. 5331–5340, 2019.

Sæmundsson, S., Hofmann, K., and Deisenroth, M. P. Meta reinforcement learning with latent variable gaussian processes. *arXiv preprint arXiv:1803.07551*, 2018.

Satorras, V. G. and Estrach, J. B. Few-shot learning with graph neural networks. In *International Conference on Learning Representations*, 2018.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320. PMLR, 2015.

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

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

van der Pol, E., Kipf, T., Oliehoek, F. A., and Welling, M. Plannable approximations to mdp homomorphisms: Equivariance under actions. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1431–1439, 2020.

Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W. M., Dudzik, A., Huang, A., Georgiev, P., Powell, R., et al. Alphastar: Mastering the real-time strategy game starcraft ii. *DeepMind Blog*, 2019.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Wang, T. and Ba, J. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*, 2019.

Wang, T., Liao, R., Ba, J., and Fidler, S. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Zhang, C., Bütepage, J., Kjellström, H., and Mandt, S. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):2008–2026, 2018.

Supplementary Materials

A. Limitations

In principle, we summarize three limitations in this work. (1) Even though MBMRL can achieve both data efficiency and fast skill transfer in a theoretical sense, the instability of policy performance, including GSSM and other models, is unavoidable due to the bias of dynamics models. This is a long-standing challenge in the model-based domain. (2) The use of message passing enables more flexible representations of node features and improves the predictive performance of GSSM in dynamical systems. But the performance might be restricted by the setting that a fully connected graph is constructed as default. So future investigations can be incorporating sparse priors into graph structures to further improve generalization. (3) Dyna-style is used in training GSSM, so a potential limitation can be additional time required to configure appropriate hyper-parameters, which decide when to collect new transitions and where to stop policy optimization. Future work can focus on designing heuristic rules for an optimal parameter setting of Dyna-style training.

Table 3. Summary of Typical MBMRL Models. Encoders are for dynamics models. Policy search strategies include model predictive control (MPC), policy gradient (PG) methods and amortized policy search (APS). As for fast adaptation, we consider whether this step is directly included in learning dynamics models or policy search.

MBMRL	Encoders	Policy Search	Dynamics Model	Fast adaptation
L2A	NULL	MPC	MAML	DM
MLSM-v0	$q_\phi(z\vert\texttt{MeanPool}([x_c, y_c]))$	PG	LVM	DM
MLSM-v1	$q_{\phi_1}(z\vert\texttt{MeanPool}([x_c, y_c]))$ $f_{\phi_2}(z_t\vert\texttt{Attn}([x_c, y_c], x_t))$	PG	LVM	DM
GSSM	$q_{\phi_2}(z_c\vert\texttt{GNN}([x_c, y_c]))$ $q_{\phi_1}(z_t\vert\texttt{GNN}([x_c, y_c], x_t))$	PG	LVM	DM
GSSM+APS	$q_{\phi_2}(z_c\vert\texttt{GNN}([x_c, y_c]))$ $q_{\phi_1}(z_t\vert\texttt{GNN}([x_c, y_c], x_t))$	APS	LVM	DM/PS

B. Other Discussions

According to feedback from other reviewers, we summarize frequently asked questions and add explanations in this part. This is to make our work clearer to potential readers.

(1) Possibility to combine Graph Neural Net (GNN) modules with meta model-free RL methods.

In our work, a combination of GNN latent variable and meta model-free methods is not applicable for an ablation experiment. That is because (1) the input and output for GNN modules cannot be accordingly specified (in model-based settings, the input and output are respectively $[s, a]$ and Δs; in model-free settings, the input is s for policies). (2) the optimization objectives for GNN modules differ a lot in model-based (Maximize the likelihood of dynamics prediction in GNN related

Table 4. Mean Square Errors (MSEs) in Meta-testing Tasks using Learned Dynamics Models. (For each testing task, 50 episodes are sampled to average. Figures in brackets are standard deviations across testing tasks, with bold ones the best.)

ENV	GSSM	M-DPILCO	MLSM-V0	MLSM-V1	L2A
CART-POLE	**0.0296($\pm$0.042)**	0.0475($\pm$0.051)	0.0626($\pm$0.081)	0.0310($\pm$0.036)	0.0397($\pm$0.04)
ACROBOT	**0.0024($\pm$0.0019)**	0.0030($\pm$0.0029)	0.004($\pm$0.0042)	**0.0024($\pm$0.0021)**	0.0039($\pm$0.0017)
H-CHEETAH	**0.530($\pm$0.22)**	0.678($\pm$0.14)	0.533($\pm$0.14)	0.636($\pm$0.14)	0.785($\pm$0.084)
S-HUMANOID	1.78($\pm$0.13)	1.9($\pm$0.15)	2.0($\pm$0.16)	**1.75($\pm$0.15)**	2.364($\pm$0.078)

Table 5. Parameter Scales of Context-based Dynamics Models in different tasks. It can be found GSSM and MLSM-v0 have the same model complexity while MLSM-v1 has more parameters in meta dynamics models.

	MLSM-v0	MLSM-v1	GSSM
Cart-Pole	$8.9 * 1e4$	$9.3 * 1e4$	$8.9 * 1e4$
Acrobot	$8.1 * 1e5$	$8.4 * 1e5$	$8.1 * 1e5$
H-Cheetah	$2.1 * 1e5$	$2.3 * 1e5$	$2.1 * 1e5$
S-Humanoid	$2.3 * 1e5$	$2.6 * 1e5$	$2.3 * 1e5$

modules) and model-free cases (Maximize the expected rewards in GNN related modules). (3) we do not find an appropriate GNN related meta model-free RL baseline, and the extension is non-trivial to design.

(2) Technical summary of context-based meta model-based RL methods.

Comparisons between our developed graph structured surrogate model and other dynamics models are summarized in Table (3). We mainly focus on methods to enable fast adaptation in both dynamics models and policy networks.

(3) Performance comparison to other meta model-based RL algorithms with non-GNN encoders.

In Section 5.1, non-GNN encoders correspond to NPs in MLSM, where a mean reduction is already used to obtain the encoded latent variable. Recurrent encoders are improper in our settings since the randomly sampled transitions from the memory buffer to identify the dynamical system are not completely sequential in dataset. But the use of Recurrent encoders are more effective when the transitions in the dynamics buffer are collected and stored in an ordered way. The model benefits from the sequential information. In this case, CaDM (Lee et al., 2020) can achieve SOTA performance in the domain.

(4) Performance comparison to other existing meta model-free RL methods, e.g. RL2 (Duan et al., 2016).

See Section 5.1, PE-PPO follows the same implementation in PEARL (Rakelly et al., 2019) except that PPO is used in policy optimization. This is to make sure all policy optimization methods are consistent in experimental analysis. We also try PEARL in model-free experiments with 1x volume of samples, but results are poorer than used baselines. Meanwhile, please refer to learning curves of other model-free meta RL papers with 1x volume of samples, e.g. FOCAL (Li et al., 2020) and MBML (Li et al., 2019), conclusions are: with limited episodes (1x samples), model-free ones, including RL2 or Learn2Learn, work far worse than model-based baselines illustrated in our papers. This means the selection of model-free meta RL baselines does not influence the comparison results and this is due to the performance bottleneck of model-free ones with limited training episodes.

C. GSSM Modules in PyTorch

Our work GSSM is built up on structures of GNNs (Kipf & Welling, 2016; Satorras & Estrach, 2018; Wang et al., 2018). But unlike vanilla GNNs, we try to learn the normalized graph Laplacian in modeling. Also, GSSMs make use of message passing in GNN modules and transform the learn node representations into target transition latent variable z_t and the prior global latent variable z_c for a MDP.

Here we rewrite the graph convolutional operation in the form of a node feature matrix $\mathcal{V}_C$, which is easier to implement in programming. The graph convolution operator $\circ$ over any data point x_t can be defined,

$$f(\mathcal{V}_C) = \sigma\left(D^{-1}L[\mathcal{V}_C\mathcal{W}]\right) \tag{12a}$$

$$\mathcal{G}(x_t) \circ f = \sum_{i\in\mathcal{O}_t} f(v_i)\texttt{sim}(x_t, x_i) \tag{12b}$$

where $\mathcal{V}_C$ is the feature matrix of the context points and $\mathcal{W}$ denotes a trainable layer matrix. The weight parameter is $\texttt{sim}(x_t, x_i)$ and $f(v_i)$ is the embedding of a node in the graph $\mathcal{G}$ after message passing processes from its neighbors $\mathcal{O}_t$. The graph Laplacian matrix L reveals the connection relationship, where D is a diagonal matrix to normalize the row elements in Eq. (12). All of these correspond to the left side of Fig. (1). The equivalent element-wise graph operation can be found in Eq. (2). The following PyTorch code is about our defined graph convolution operations and the induced meta dynamics model.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.nn.parameter import Parameter
from torch.autograd import Variable


class GC_Net(nn.Module):
    '''
    Graph convolutional layer for message passing: graph Laplacian matrix construction.
    Feature transformation used only once.
    Message passing across context points [x_c,y_c] and from context points to target
    points [x_t,y_t].
    '''
    def __init__(self,dim_x,dim_y,dim_emb_x,dim_lat,dim_h_lat,num_h_lat,
        trans_xy:bool=True,requires_grad:bool=False):
        super(GC_Net,self).__init__()

        self.dim_x=dim_x
        self.dim_y=dim_y
        self.dim_emb_x=dim_emb_x
        self.dim_lat=dim_lat
        self.dim_h_lat=dim_h_lat
        self.num_h_lat=num_h_lat
        self.trans_xy=trans_xy
        self.requires_grad=requires_grad

        if self.requires_grad:
            self.beta=Parameter(torch.Tensor(1).uniform_(0,1),requires_grad=self.
    requires_grad).cuda()
        else:
            self.beta=Variable(torch.ones(1),requires_grad=self.requires_grad).cuda()

        self.fc_x=nn.Sequential(nn.Linear(self.dim_x, self.dim_emb_x, bias=False)).cuda()

        self.trans_modules=[]
        if self.trans_xy :
            self.trans_modules.append(nn.Linear(self.dim_x+self.dim_y, self.dim_h_lat,
    bias=False))
        else:
            self.trans_modules.append(nn.Linear(self.dim_y, self.dim_h_lat, bias=False))
            self.trans_modules.append(nn.LayerNorm(self.dim_h_lat))
        self.trans_modules.append(nn.ReLU())
        for i in range(self.num_h_lat):
            self.trans_modules.append(nn.Linear(self.dim_h_lat, self.dim_h_lat))
            self.trans_modules.append(nn.ReLU())
        self.trans_modules.append(nn.Linear(self.dim_h_lat, self.dim_lat))

        self.trans_net=nn.Sequential(*self.trans_modules).cuda()


    def emb_aggregator(self,h_context,aggre_dim):
        '''
        aggregation embeddings from the context points.
        '''
        h_aggre=torch.mean(h_context,dim=aggre_dim)

        return h_aggre


    def forward(self,x_context,x_target,y_context,
                self_addition=True,whether_l2norm=False):
        '''
        construct GL based on pairwise similarities.
        x_context-->shape [num_task,num_c_points,dim_x].
        '''
        assert x_context.dim()== 3

        x_emb_c, x_emb_t = self.fc_x(x_context), self.fc_x(x_target)

        x_emb_cn, x_emb_tn = torch.norm(x_emb_c, p=2, dim=-1, keepdim=True).detach(),
    torch.norm(x_emb_t, p=2, dim=-1, keepdim=True).detach()
        x_emb_c, x_emb_t = x_emb_c.div(x_emb_cn.expand_as(x_emb_c)),\
                        x_emb_t.div(x_emb_tn.expand_as(x_emb_t))


        b_mask=(zero_diag_mask(x_emb_c)).cuda()

        c_inner_prod=self.beta*torch.matmul(x_emb_c,x_emb_c.transpose(1,2))
        c_n_prod=F.softmax(c_inner_prod,dim=-1)

        mask_c_n_prod=torch.mul(c_n_prod,b_mask)
        rand_mat=F.normalize(mask_c_n_prod,p=1,dim=-1)
        b_id_mat=batch_eye_mat(x_emb_c)
        g_lap_mat=rand_mat+b_id_mat

        # compute normalized dot products between x_c and x_t
        t_inner_prod=self.beta*torch.matmul(x_emb_t,x_emb_c.transpose(1,2))
        t_n_prod=F.softmax(t_inner_prod,dim=-1)

        # formulate the latent representation for each context data point
        if self.trans_xy:
            context_mat=torch.cat((x_context,y_context),dim=-1)
        else:
            context_mat=y_context
        emb_context=self.trans_net(context_mat)

        if self_addition:
            c_n_prod_unsq=g_lap_mat.unsqueeze(-1)
        else:
            c_n_prod_unsq=c_n_prod.unsqueeze(-1)
        emb_c_unsq=emb_context.unsqueeze(1).expand(-1,context_mat.size(1),-1,-1)
        c_message_pass=emb_c_unsq.mul(c_n_prod_unsq)
        c_message_aggregation=c_message_pass.sum(2)
        c2t_message_aggregation=(torch.mean(c_message_aggregation,dim=1)).unsqueeze(1)
        c2t_message_aggregation=c2t_message_aggregation.expand(-1,x_target.size(1),-1)

        # message passing from the context to the target point
        c_message_aggregation_unsq=c_message_aggregation.unsqueeze(1).expand(-1,
                                    x_target.size(1),-1,-1)
        t_n_prod_unsq=t_n_prod.unsqueeze(-1)
        t_message_pass=c_message_aggregation_unsq.mul(t_n_prod_unsq)
        t_message_aggregation=self.emb_aggregator(t_message_pass,aggre_dim=2)

        if whether_l2norm:
            c2t_message_aggregation = l2_normalization(c2t_message_aggregation)
            t_message_aggregation = l2_normalization(t_message_aggregation)

        return c_message_aggregation, c2t_message_aggregation, t_message_aggregation


def batch_eye_mat(x):

    identity_mat=(torch.eye(x.size(1))).reshape((1,x.size(1),x.size(1)))
    b_identity_mat=identity_mat.repeat(x.size(0),1,1).cuda()

    return b_identity_mat


def zero_diag_mask(x):

    identity_mat=(torch.eye(x.size(1))).reshape((1,x.size(1),x.size(1)))
    b_identity_mat=identity_mat.repeat(x.size(0),1,1)

    b_one_mat=torch.ones(x.size(0),x.size(1),x.size(1))

    b_mask=b_one_mat-b_identity_mat

    return b_mask


def l2_normalization(x_tensor):
    x_l2_norm = torch.norm(x_tensor, p=2, dim=-1, keepdim=True).detach()
    normalized_x = x_tensor.div(x_l2_norm.expand_as(x_tensor))

    return normalized_x
```

Listing 1. Message Passing Modules in the PyTorch Version.

```
class GS_DM(nn.Module):
    '''
    Learning meta dynamics model via a Graph Structured Surrogate Model.
    '''
    def __init__(self,args):
        super(GS_DM,self).__init__()

        #extract parameters from args
        self.dim_x=args.dim_x
        self.dim_y=args.dim_y

        self.dim_emb_x=args.dim_emb_x
        self.dim_lat=args.dim_lat
        self.dim_h_lat=args.dim_h_lat
        self.num_h_lat=args.num_h_lat

        self.dim_h=args.dim_h
        self.num_h=args.num_h
        self.act_type=args.act_type
        self.amort_y=args.amort_y

        #graph context embeddings
        self.gc_net=GC_Net(self.dim_x, self.dim_y, self.dim_emb_x,
                    self.dim_lat, self.dim_h_lat, self.num_h_lat).cuda()

        self.mu_net=nn.Sequential(nn.Linear(self.dim_lat, self.dim_lat)).cuda()
        self.logvar_net=nn.Sequential(nn.Linear(self.dim_lat, self.dim_lat)).cuda()

        self.mu_net_g=nn.Sequential(nn.Linear(self.dim_lat, self.dim_lat)).cuda()
        self.logvar_net_g=nn.Sequential(nn.Linear(self.dim_lat, self.dim_lat)).cuda()

        self.dec_modules=[]
        self.dec_modules.append(nn.Linear(self.dim_x+self.dim_lat, self.dim_h))
        for i in range(args.num_h):
            self.dec_modules.append(get_act(args.act_type))
            self.dec_modules.append(nn.Linear(self.dim_h, self.dim_h))
        if self.amort_y:
            self.dec_modules.append(get_act(args.act_type))
            self.dec_modules.append(nn.Linear(self.dim_h, 2*self.dim_y))
        else:
            self.dec_modules.append(get_act(args.act_type))
            self.dec_modules.append(nn.Linear(self.dim_h, self.dim_y))
        self.dec_net=nn.Sequential(*self.dec_modules).cuda()



    def get_context_idx(self,M):
```

```python
        # generate the indeces of the N context points from M points
        N = random.randint(1,M)
        idx = random.sample(range(0, M), N)
        idx = torch.tensor(idx).cuda()

        return idx


    def idx_to_data(self,data,sample_dim,idx):
        # get subset of an array
        ind_data= torch.index_select(data, dim=sample_dim, index=idx)

        return ind_data


    def reparameterization(self,mu,logvar):
        #sample some random variable from N(0,I) and derive the correspinding z-sample.
        std=torch.exp(0.5*logvar)
        eps=torch.randn_like(std)

        return mu+eps*std


    def forward(self,x_memory,y_memory,x_pred):
        c_emb, c2t_message_aggregation, gc_emb=self.gc_net(x_memory,x_pred,y_memory)
        mu=self.mu_net(gc_emb)
        logvar=self.logvar_net(gc_emb)
        mu_g=self.mu_net_g(c2t_message_aggregation)
        logvar_g=self.logvar_net_g(c2t_message_aggregation)

        z=self.reparameterization(mu, logvar)

        output=self.dec_net(torch.cat((x_pred,z),dim=-1))

        if self.amort_y:
            y_mean,y_var=output[:,:,:self.dim_y],F.softplus(output[:,:,self.dim_y:])
            return mu,logvar,mu_g,logvar_g,y_mean,y_var
        else:
            y_pred=output
            return mu,logvar,mu_g,logvar_g,y_pred
```

Listing 2. Graph Structured Surrogate Models in the PyTorch Version.

D. Evidence Lower Bound for GSSM

Here $P(D)$ denotes the distribution of state action pairs in meta-training processes, and each data point is attached with a context set $[x_c, y_c]$ (a batch of transition data points) to imply the statistics information from a task. With a Jessen's inequality and an approximate posterior $q_\phi(z_t|x_t, x_c, y_c)$, we can have evidence lower bound as follows.

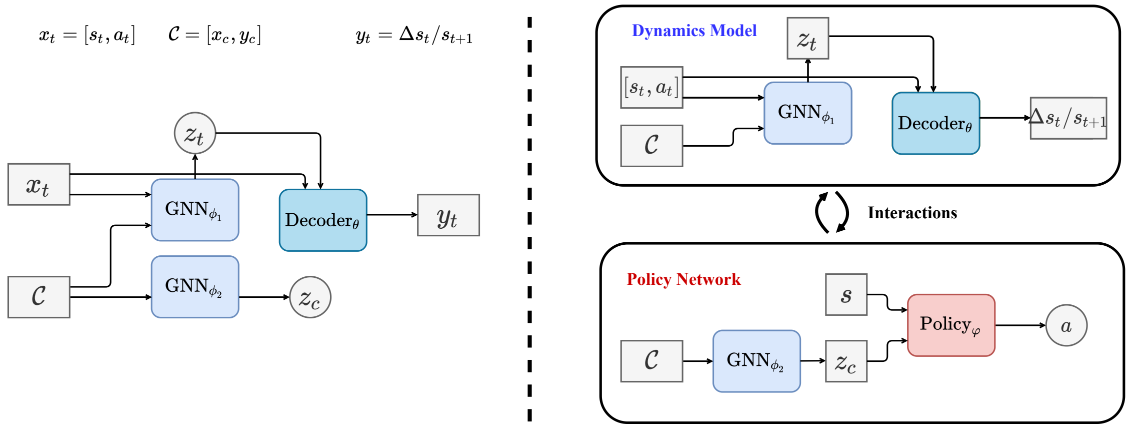
$$\begin{aligned}\mathbb{E}_{p(D)} \ln p(y_t|x_t, x_c, y_c) &= \mathbb{E}_{p(D)} \ln \mathbb{E}_{q_\phi}\Big[\frac{p(z_*)}{q_\phi(z_t|x_t, x_c, y_c)} p_\theta(y_t|x_t, z_t)\Big] \\ &\geq \mathbb{E}_{p(D)}\Bigg[\mathbb{E}_{q_\phi} \ln \big[p_\theta(y_t|x_t, z_t)\big] - \mathbb{E}_{q_\phi} \ln \Big[\frac{q_\phi(z_t|x_t, x_c, y_c)}{p(z_t)}\Big]\Bigg]\end{aligned} \tag{13}$$

By replacing the zero information prior distribution $p(z_t)$ with $q(z_t|x_c, y_c)$, we can derive the formerly mentioned ELBO.

$$\mathbb{E}_{p(D)}\Big[\ln \underbrace{p(y_t|x_t, x_c, y_c)}_{\text{intractable data likelihood}}\Big] \geq \mathbb{E}_{p(D)}\Big[\mathbb{E}_{q_{\phi_1}}\big[\ln p_\theta(y_t|x_t, z_t)\big] - D_{KL}\big[\underbrace{q_{\phi_1}(z_t|x_t, x_c, y_c)}_{\text{approximate posterior}} \parallel \underbrace{q_{\phi_2}(z_c|x_c, y_c)}_{\text{approximate prior}}\big]\Big] \tag{14}$$

Note that both the approximate prior and the posterior are learnable with a partially shared neural network in meta learning scenarios, which is similar in work (Denton & Fergus, 2018; Pertsch et al., 2020; Garnelo et al., 2018b). And the learned prior $q_{\phi_2}(z_c|x_c, y_c)$ can be viewed as a summary of context points, which is further used to help induce amortized policies $\pi_\varphi(a|s, z_c)$. For more details on encoding relationship between these context points and target points, refer to Fig. (1).

Besides, the number of context points is a random number smaller than the batch size in meta training dynamics models, which shares the same setting as Neural Processes. As mentioned in the Main Paper, a fully connected graph is built among context points, and the graph Laplacian matrix is learned based on Eq. (2.a). Here we treat all the context points as the neighborhood $\mathcal{O}_t$ of a target point x_t. The way of constructing fully connected graphs is a limitation for GSSMs, and future work can be discovery of optimal graph structures to further improve performance.



(a) Computational Graph in Meta-training Dynamics Models

(b) Meta-training Amortized Policies

Figure 5. Computational Graphs of GSSM in Meta-training Processes. Note that amortized policy search is used here. On the **Left** side of the Figure: it describes the connections of variables in neural networks for meta dynamics models. On the **Right** side of the Figure: it illustrates the process in meta model-based policy search (Note that the reparameterization trick is used here and sampled latent variables z_t and z_c are deterministic, denoted by squares).

E. Computational Graphs and Detailed Descriptions

Here we add more explanations about our developed GSSM and amortized policy search. This section corresponds to Fig. (1). Especially, computational processes are displayed in Fig. (5).

On the **Left** of Fig. (5), the $\texttt{GNN}_{\phi_2}$ module is used to summarize the task using $q_{\phi_2}(z_c)$ from the context points $\mathcal{C}$, while the $\texttt{GNN}_{\phi_1}$ module is used to learn state-action pair x_t dependent latent variables $q_{\phi_1}(z_t)$. As mentioned in the Main paper, these two modules share part of parameters. The concatenation of x_t and sampled z_t is input into the $\texttt{Decoder}_\theta$ module to predict the transited state s_{t+1} or state difference Δs_t.

On the **Right** of Fig. (5), this depicts the process in learning amortized policies (Mainly refer to the loop iterations `step (6)-(10)` in Algorithm (1)). Given a batch of MDPs, context points $\mathcal{C}$ are sampled using a uniform random policy and input into $\texttt{GNN}_{\phi_2}$ to formulate task-specific latent variables. Note that the reparameterization trick is used in this process, which means we sample $\epsilon \sim \mathcal{N}(0, I)$ during the loop of iterations to formulate $z_t = \mu_t + \Sigma_t^{-\frac{1}{2}}\epsilon$ for state-action pairs $x_t = [s_t, a_t]$. This operation can be interpreted as sampling MDPs from the posterior in posterior sampling (Osband et al., 2013). Similarly, we sample $\epsilon \sim \mathcal{N}(0, I)$ to formulate $z_c = \mu_c + \Sigma_c^{-\frac{1}{2}}\epsilon$ and get this sample value retained in the loop iterations (In Actor-Critic cases, the sampled value also takes part in value function approximators). During the process, these task-specific policies interact with a collection of sampled MDPs to seek optimal results like that in BOSS (Asmuth et al., 2009), and this corresponds to find optimal policies w.r.t sampled MDPs in posterior sampling. In the next loop of iterations, new transitions are collected using the amortized policy and the dynamics model is retrained to update the posterior.

F. Policy Gradient Estimates in Amortized Policy Search

Due to page limits, we provide more details in actor critic cases. Just as revealed in Fig. (1), the right one is to show amortized policy search in a collection of approximate dynamics models. Estimates of policy gradients are formulated and these details will tell readers how our amortized policies are learned in MBMRL. All of these correspond to `step (10)` in Algorithm (1).

Note that parameters φ in our amortized policies $\pi_\varphi(a|[s,z_c])$ are optimized in developed dynamics models. Dynamics models as GSSMs consist of parameters θ and $\phi=[\phi_1,\phi_2]$, and we denote a sampled approximate dynamics model for one task as $\hat{\mathcal{M}}$. As for a distribution of synthetic trajectories τ from a learned dynamics model $\hat{\mathcal{M}}$, we use $p(\tau|\hat{\mathcal{M}};\varphi,\phi)$ to define, where q_{ϕ_2} encodes contextual information from $[x_c,y_c]$ for different tasks. Given a sampled trajectory τ from an approximate dynamics model $\hat{\mathcal{M}}$, we can decompose it according to the Markov property.

$$p(\tau|\hat{\mathcal{M}};\varphi,\phi)=p(s_0)\prod_{t=0}^{T-1}\left[p(s_{t+1}|s_t,a_t,z_t)\pi_\varphi(a_t|[s_t,z_c])\right] \tag{15}$$

Besides, the proposed amortized policy search strategy can be combined with any other dynamics model with contextual latent variables, not limited to Graph Structured Surrogate Models in our paper (e.g. NP (Garnelo et al., 2018b; Galashov et al., 2019), it is also available to combine with our developed policy search strategy).

We consider *actor-critic* frameworks, where a value function is also conditioned on the latent variable (Refer to the right side of Fig. (1). Hence, two objectives, namely value function approximation and policy optimization, are involved in this setting.

$$\mathcal{L}_{\text{C}}(\hat{\varphi})=\mathbb{E}_{\hat{\mathcal{M}}\sim p(\hat{\mathcal{M}};\theta,\phi)}\mathbb{E}_{\substack{(s,a,s',r)\sim\mathcal{B}(\hat{\mathcal{M}})\\ z_c\sim q_\phi(z_c|[x_c,y_c])}}\left[Q_{\hat{\varphi}}([s,z_c],a)-(r+\gamma V_{\tilde{\varphi}}([s',z_c]))\right]^2 \tag{16}$$

The value function approximation objective is Eq. (16), where $\mathcal{B}(\hat{\mathcal{M}})$ is a batch of synthetic transition samples from $\hat{\mathcal{M}}$ and $\hat{\varphi}$ is the parameter of value function approximators. The critic optimization process is inside policy search during meta-training processes via gradient updates. The actor optimization objective can be found in the main paper.

To optimize policy functions as that in Eq. (10), we can utilize a likelihood ratio trick (Williams, 1992) and compute the derivative in transitions as follows.

$$\ln p(\tau|\hat{\mathcal{M}};\varphi,\phi)=\ln p(s_0)+\sum_{t=0}^{T-1}\left[\ln p(s_{t+1}|s_t,a_t,z_t))+\ln\pi_\varphi(a_t|[s_t,z_c])\right] \tag{17}$$

$$\nabla_\varphi\ln p(\tau|\hat{\mathcal{M}};\varphi,\phi)=\nabla_\varphi\ln\cancel{p(s_0)}+\sum_{t=0}^{T-1}\nabla_\varphi\left[\ln\cancel{p(s_{t+1}|s_t,a_t,z_t)}+\ln\pi_\varphi(a_t|[s_t,z_c])\right] \tag{18}$$

Based on Eq. (18), we formulate the estimated policy gradient in Eq. (19), and a modified PPO (Schulman et al., 2017) is used as an instantiation to implement in our settings.

$$\begin{aligned}\nabla_\varphi\mathcal{J}(\varphi)&=\mathbb{E}_{\hat{\mathcal{M}}\sim p(\mathcal{M};\theta,\phi)}\left[\int\nabla_\varphi p(\tau|\hat{\mathcal{M}};\varphi,\phi)\mathcal{R}(\tau)d\tau\right]\\&=\mathbb{E}_{\hat{\mathcal{M}}\sim p(\mathcal{M};\theta,\phi)}\mathbb{E}_{\tau\sim p(\tau|\hat{\mathcal{M}};\varphi,\phi)}\left[\nabla_\varphi\ln p(\tau|\hat{\mathcal{M}};\varphi,\phi)\mathcal{R}(\tau)\right]\\&=\mathbb{E}_{\hat{\mathcal{M}}\sim p(\mathcal{M};\theta,\phi)}\mathbb{E}_{\tau\sim p(\tau|\hat{\mathcal{M}};\varphi,\phi)}\left[\sum_{t=0}^{T-1}\nabla_\varphi\ln\pi(a_t|[s_t,z_c])\right]\cdot\left[\sum_{t=0}^{T-1}r(s_t,a_t,s_{t+1})\right]\end{aligned} \tag{19}$$

Meanwhile, the policy gradient in the form of advantage functions is derived in Eq. (20),

$$\nabla_{\varphi}\mathcal{J}(\varphi) = \mathbb{E}_{\substack{\hat{\mathcal{M}}\sim p(\hat{\mathcal{M}};\theta,\phi)\\ \tau\sim p(\tau|\hat{\mathcal{M}};\varphi,\phi)}}\Big[\sum_{t=0}^{T-1}\nabla_{\varphi}\ln\pi(a_t|[s_t,z_c])\cdot A_t([s_t,z_c],a_t)\Big] \tag{20}$$

where $A_t([s_t,z_c],a_t)$ is an advantage function, mostly written as the difference between a cumulative reward term and a baseline term $A_t([s_t,z_c],a_t)=\sum_{t'=t+1}^{T-1} r([s_{t'},z_c],a_{t'}) - b_t([s_t,z_c])$.

Similar to the term on the right side of Eq. (10), the corresponding Monte Carlo estimate of the policy gradient can be easily obtained from Eq. (20), and we skip this step in this section.

G. Experimental Settings and Training Details

In this section, we provide information about environments and give more details in meta training/testing processes. Especially, the updated code file can be found in (`https://github.com/hhq123gogogo/GSSM_APS`). Only one layer graph convolution is used in general settings.

G.1. Environmental Details

MBMRL Tasks. Here we describe meta reinforcement learning tasks in this paper. The Cart-Pole environment can be found in the link[2] here. The Acrobot is based on open-ai gym[3]: with continuous states $[\theta_1,\theta_1',\theta_2,\theta_2']$ as angles and instant angle velocities, the goal is to sequentially select an action from $\{-1,0,+1\}$ (respectively Right Torque, No Torque, Left Torque) to reach the height above the top of the pendulum as early as possible. Half-Cheetah/Slim-Humanoid are from a Mujoco package[4] and both are to conduct locomotion tasks. Generations of diverse Cart-Pole/Acrobot environments have been introduced in the main paper. As for Cart-Pole tasks, 50 unseen tasks are sampled from simulators for meta testing and each task is with 50 episodes in evaluation (refer to Fig. (2)). As for Acrobot tasks, 33 unseen tasks are sampled from simulators for meta testing and each task is with 50 episodes in evaluation (refer to Table (2)/(4)). As for configurations of Half-Cheetah/Slim-Humanoid environments, we generate the Meta-training MDPs via the combination of the mass re-scaled coefficient in the list $\{0.8,0.9,1.0,1.1,1.2\}$ and the damping coefficient in the list $\{0.8,0.9,1.0,1.1,1.2\}$, while those hyper-parameters for Meta-testing phases are $\{0.85,0.95,1.05,1.15\}$ for both mass-rescaled and damping coefficients. As a result, totally 16 unseen MDPs are generated by the Cartesian of mass coefficients and damping coefficients for meta-testing processes (refer to Table (4) and Table (2)/(4)).

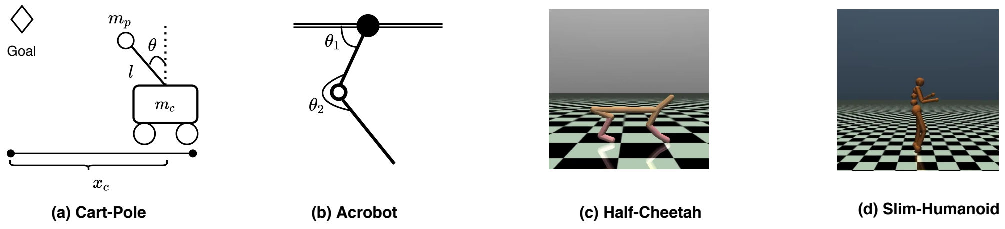


Figure 6. Fundamental Environments used in Meta Model-based Reinforcement Learning Experiments.

Reward Descriptions. Besides, reward functions are listed here (refer to Table (6)). More details are as follows. In Cart-Pole environments, d in a reward function measures the square of the distance between the pole's end point and its goal, and hyper-parameter $\sigma_c = 0.25$. In Acrobot environments, the list of parameters $\{l_1,l_2,\theta_1,\theta_2\}$ corresponds to Fig. (6) in terms of meanings in a reward function. In Half-Cheetah environments, x_t is the notation of the x-coordinate in the Half-Cheetah agent at time slot index t, ∇_t is time difference in dynamics (the resulted ratio is the speed of agent.) and a_t is

[2]https://github.com/BrunoKM/deep-pilco-torch
[3]https://gym.openai.com/
[4]http://www.mujoco.org/

Table 6. Reward Functions in Related Environments.

ENV	REWARD FUNCTIONS	HORIZON	CONTROL
CART-POLE	$1-\exp{(-\frac{\|d^2\|}{\sigma_c^2})}$	25	CONTINUOUS
ACROBOT	$\text{BOOL}(-l_1\cos{(\theta_1)}-l_2\cos{(\theta_1+\theta_2)}-l_1)$	200	DISCRETE
H-CHEETAH	$\frac{x_{t+1}-x_t}{\nabla t}-0.1*\|a_t\|_2^2$	1000	CONTINUOUS
S-HUMANOID	$\frac{50(x_{t+1}-x_t)}{3\nabla t}-0.1*\|a_t\|_2^2+5.0*\text{BOOL}(1.0\leq x_{t,h}\leq 2.0)$	1000	CONTINUOUS

the action performed instantly. In Slim-Humanoid environments, notations are similar to those in Half-Cheetah and $x_{t,h}$ in reward functions refer to the instant torso's height. Horizons of trajectories as well as types of action spaces can also be found in Table (6).

Learning Parameterized Policies in the Background of MBMRL

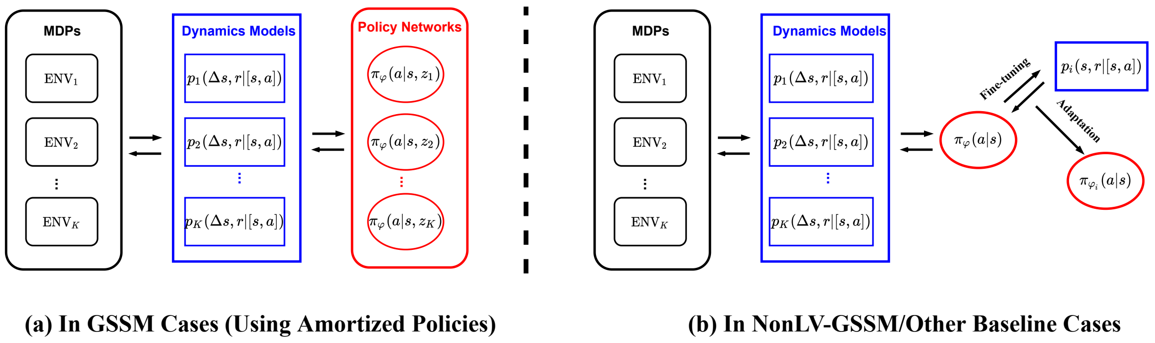


Figure 7. Meta Model-based Policy Search used in Models. Note that reducing adaptation time in policies is the first priority in this work. On the **Left**, amortized policies are used and latent variables are to specify different tasks. On the **Right**, the meta-trained policy is not conditioned on latent variables and needs to be adapted to respective tasks, which consumes additional time.

G.2. Training Details

Data Preprocessing. In Acrobot tasks, the output of dynamics models is the next or transited state ($x=[s,a]$, $y=s'$). In other tasks, the output of dynamics models is the difference of the next state and the current state ($x=[s,a]$, $y=\Delta s$). For the input of dynamics models, it is the state-action pair in all environments. For Half-Cheetah/Slim-Humanoid environments, standardization is required for both the input and the output of dynamics models in meta training processes.

More Details in Policy Search. In Cart-Pole Swing-Up environments, Back-Propagation Through Time (BPTT) is used in model-based policy search and the policy network parametrized with a radial basis function follows that in (Gal et al., 2016),for GSSM+APS, a latent variable is concatenated with the state variable as the input. In Acrobot/Half-Cheetah/Slim-Humanoid, we combine PPO with the learned dynamics model (We also perform additional trials in BPTT strategies but this kind of model-based policy search suffers from gradient exploding in practice), and a direct combination of model-based and model-free RL algorithms in meta-learning leads to stable training.

Besides, meta-trained policies in MLSM-v0/MLSM-v1/M-DPILCO require additional policy gradient updates in separate dynamics models of tasks and these are up to tasks based on our trials: for Cart-Pole/Acrobot, five trajectories are enough to fine-tune these policies, and for Half-Cheetah/Slim-Humanoid one trajectory is enough to fine-tune these policies. Such adaptation in MLSM-v0, MLSM-v1 and M-DPILCO consumes additional time when the learned policy is implemented in unseen environments.

The pipeline is reflected in Fig. (7). Traditional model predictive control strategies are prohibitively expensive in implementations, costing much more time with lower efficiency in high dimensional action space. Since related work employing parameterized policies in MBMRL remains limited, our proposed method can be deemed as a preliminary exploration.

Further Descriptions in Fig.s/Tables. Here we add more descriptions on Cart-Pole, where authors can follow the implementations in the work[5], and the state-of-art performance using Deep-PILCO is about -0.6 in episodes for a single task. We also try DR-PPO and PE-PPO in Cart-Pole tasks with more than 10x required time steps in training, but the resulted performance in testing is far worse than MBMRL ones and we guess the PPO algorithm here cannot well handle planning with short horizons (Other referred model-free results can be found in (Lillicrap et al., 2016). In addition, the estimated required samples of all MBMRL baselines for Cart-Pole are even 2x less than model-free ones (Lillicrap et al., 2016; Gal et al., 2016) to train in one single MDP.).

In Fig. (2.(a)/(b)/(d)), each `iter` in x-axis indicates that a new trajectory is sampled to update the dynamics buffer, the batch size of samples in dynamics memory buffer is 100, the default epoch in training dynamics models is 5 in each `iter`, and the Cart-Pole environment changes every every 10 `iter`. Meanwhile, for each `iter`, 25 trajectories are sampled using the updated policy to average results for evaluating the performance in the trained task (this process results in learning curves).

Fig. (4) keeps track of meta-training performance using MBMRL algorithms in Acrobot/H-Cheetah/S-Humanoid, and dynamics of MDPs change with iterations (the batch numbers of tasks are 1 for Acrobot/H-Cheetah and 3 for S-Humanoid). In Acrobot, the batch size of samples in dynamics memory buffer is 100, the default epoch in training dynamics models is 20 in each `iter`, and 15 trajectories are sampled for evaluation to show performance on learning curves. In H-Cheetah, the batch size of samples in dynamics memory buffer is 1000, the default epoch in training dynamics models is 20 in each `iter`, and 15 trajectories are sampled for evaluation to show performance on learning curves. In S-Humanoid, the batch size of samples in dynamics memory buffer is 1000, the default epoch in training dynamics models is 10 in each `iter`, and 15 trajectories are sampled for evaluation to show performance on learning curves. And every fixed number of iterations, MDPs in meta-training are resampled (for Acrobot, every 3 `iters`; for H-Cheetah, every 3 `iters`; for S-Humanoid, every 1 `iter`).

Table (4) and Table (1)/(2) summarize the meta-testing results over unseen MDPs. We collect the rewards in each task using these models to obtain the average results in each task and then report average rewards over meta testing tasks in the table. Some additional explanations are as follows. In meta-testing tasks of Cart-Pole, contextual latent variables in GSSM/MLSM-v0/MLSM-v1 are computed after transitions of two trajectories (50 transition steps) are aggregated. In meta-testing tasks of Acrobot, contextual latent variables in GSSM/MLSM-v0/MLSM-v1 are computed after transitions of a one-sixth trajectory (50 transition steps) are aggregated. In meta-testing tasks of Half-Cheetah/Slim-Humanoid, contextual latent variables in GSSM/MLSM-v0/MLSM-v1 are computed after transitions of a half trajectory (500 transition steps) are aggregated.

Meanwhile meta-training processes in model-free meta reinforcement learning are recorded in Fig. (4) . All of these are trained with Adam optimizers and learning rates are 5e-4 in default.

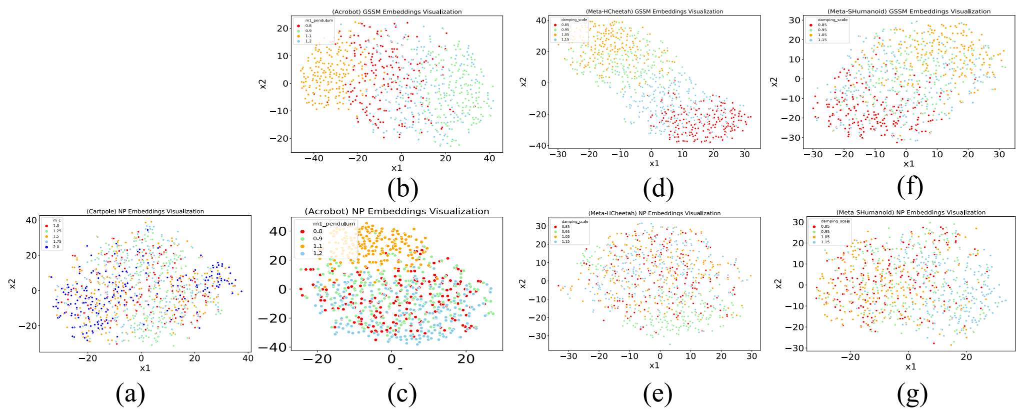


Figure 8. More Visualized Results of Latent Variables in Meta Trained Models. (a) is the plot of sampled latent variables with a pole mass as 1.0 and cart masses as $\{1.0, 1.25, 1.5, 1.75, 2.0\}$ in CartPole. Similarly, (b)/(c) are results by varying the pendulum1's mass with pendulum0's mass fixed to be 1.0 in Acrobot. Similarly, (d)/(e) are results by varying damping scale values with mass scale fixed to be 1.0 in Meta-HCheetah. Similarly, (f)/(g) are results by varying damping scale values with mass scale fixed to be 1.0 in Meta-SHumanoid.

[5]https://github.com/BrunoKM/deep-pilco-torch

Table 7. Neural Network Structure of MBMRL Models. The transformations in the table are linear, followed with ReLU activation mostly. As for MLSM-v1, the encoder network is doubled in the table since there exists a local variable for prediction.

NP Models	Encoder	Decoder
MLSM-v0/v1	$[dim\_x, dim\_y] \mapsto \underbrace{dim\_latxy \mapsto dim\_latxy}_{n\ times}$ $dim\_latxy \mapsto dim\_lat.$	$[dim\_x, (2*)dim\_lat] \mapsto \underbrace{dim\_h \mapsto dim\_h}_{m\ times}$ $dim\_h \mapsto dim\_y$
GSSM	$[dim\_x, dim\_y] \mapsto \underbrace{dim\_latxy \mapsto dim\_latxy}_{n\ times}$ $dim\_x \mapsto dim\_latx;$ $[dim\_latx, dim\_laty] \mapsto dim\_lat.$	$[dim\_x, dim\_lat] \mapsto \underbrace{dim\_h \mapsto dim\_h}_{m\ times}$ $dim\_h \mapsto dim\_y$

Table 8. Neural Network Structure in Meta Policy Networks. For Back-propagation Through Time (BPTT) and Actor-Critic Policy Gradient Algorithms, neural architectures are different. ReLU is used as an activation function. Soft-max is used in the output of Actor Networks in the discrete control.

Policy Search	Neural Architectures
BPTT	$[dim\_obs]/[dim\_obs, dim\_lat] \mapsto \underbrace{dim\_ph \mapsto dim\_ph}_{n_p\ times}$ $dim\_ph \mapsto dim\_act.$
AC-PG (PPO)	$[dim\_obs]/[dim\_obs, dim\_lat] \mapsto \underbrace{dim\_ph \mapsto dim\_ph}_{n_{pc}\ times} \mapsto dim\_act$ (Actor Network) $[dim\_obs]/[dim\_obs, dim\_lat] \mapsto \underbrace{dim\_ph \mapsto dim\_ph}_{n_{pa}\ times} \mapsto 1$ (Critic Network).

H. Neural Architectures and Parameter Settings

Here neural architectures in meta dynamics models are listed in Table (7). These architectures are shared across all implemented tasks in the paper. And one layer graph encoding is enough to guarantee performance in our GSSM implementations for all experiments. For Meta-DPILCO, neural architectures resemble that in the table except that encoders for latent variables are removed and dropout modules are integrated in each layer. In Cart-Pole environments, parameters in Table (7) are $\{n = 2, dim\_latxy = 32, dim\_lat = 16, m = 2, dim\_h = 200\}$. In Acrobot environments, parameters in Appendix Table (7) are $\{n = 2, dim\_latxy = 32, dim\_lat = 16, m = 5, dim\_h = 400\}$. In Mujoco environments, parameters in Appendix Table (7) are $\{n = 2, dim\_latxy = 32, dim\_lat = 16, m = 5, dim\_h = 400\}$. Also note that in our implementations, we set $dim\_lat = 8$ for GSSM in Cart-Pole/Acrobot/H-Cheetah because lower dimensional information bottlenecks are more compact and help amortized policy search, while for models using non-latent variable conditioned policies better results are achieved with information bottleneck $dim\_lat = 16$.

As for meta policy networks or latent variable conditioned policy networks (used in GSSM), we adopt the ordinary ones and these are listed in Table (8). In Cart-Pole environments, parameters in Table (8) are $\{n_p = 1, dim\_ph = 50\}$. In Acrobot environments, parameters in Appendix Table (8) are $\{n_{pa} = 1, dim\_ph = 128, n_{pc} = 1\}$. In Half-Cheetah environments, parameters in Table (8) are $\{n_{pa} = 1, dim\_ph = 128, n_{pc} = 1\}$. In model-free meta reinforcement learning scenarios, the contextual encoder is permutation invariant the same with that used in MLSM-v0, and the optimization objectives follow those in PEARL (Rakelly et al., 2019). Besides, we implement the vanilla version of PEARL with the same training sample volume as that in MBMRL but find the results are inferior to mentioned model-free baselines.

I. Computing Devices and Required Platforms

Throughout the work, we run experiments in a GTX 1080-Ti GPU, and Pytorch[6] is used in implementations.

[6]https://pytorch.org/