PARTIAL MODELS FOR BUILDING ADAPTIVE MODEL-BASED REINFORCEMENT LEARNING AGENTS

Safa Alver[1,3], **Ali Rahimi-Kalahroudi**[2,3], **Doina Precup**[1,3,4]
[1]McGill University, [2]University of Montreal, [3]Mila Quebec AI Institute, [4]Google DeepMind
safa.alver@mail.mcgill.ca, ali-rahimi.kalahroudi@mila.quebec, dprecup@cs.mcgill.ca

ABSTRACT

In neuroscience, one of the key behavioral tests for determining whether a subject of study exhibits model-based behavior is to study its adaptiveness to local changes in the environment. In reinforcement learning, however, recent studies have shown that modern model-based agents display poor adaptivity to such changes. The main reason for this is that modern agents are typically designed to improve sample efficiency in single task settings and thus do not take into account the challenges that can arise in other settings. In local adaptation settings, one particularly important challenge is in quickly building and maintaining a sufficiently accurate model after a local change. This is challenging for deep model-based agents as their models and replay buffers are monolithic structures lacking distribution shift handling capabilities. In this study, we show that the conceptually simple idea of *partial models* can allow deep model-based agents to overcome this challenge and thus allow for building locally adaptive model-based agents. By modeling the different parts of the state space through different models, the agent can not only maintain a model that is accurate across the state space, but it can also quickly adapt it in the presence of a local change in the environment. We demonstrate this by showing that the use of partial models in agents such as deep Dyna-Q, PlaNet and Dreamer can allow for them to effectively adapt to the local changes in their environments.

1 INTRODUCTION

Recent studies have shown that modern model-based reinforcement learning (MBRL) agents display poor signs of adaptivity to the local changes in their environments (Van Seijen et al., 2020; Wan et al., 2022), despite this being a key behavioral characteristic of model-based biological agents (Daw et al., 2011). The analysis of Wan et al. (2022) reveals that the main reason for this lack of adaptivity is because of the agent's inability in building and maintaining a sufficiently accurate model after a local change. This is a challenge for modern model-based agents as their models and replay buffers are monolithic structures lacking distribution shift handling capabilities. More specifically, in deep model-based agents the data that is used in updating the agent's model is stored in a single replay buffer and thus, in the face of a local change, leads to problems like (i) the interference of the old and new data and (ii) the forgetting of the old-but-relevant data (Wan et al., 2022; Rahimi-Kalahroudi et al., 2023). Moreover, the data that is used in the updates is sampled in a random fashion, which leads to problems like the agent ending up with a biased model. Finally, the model that is used in updating the agent's policy is a single model and thus leads to problem like the inability to perform to-the-point updates for quick adaptation in the face of a local change.

To address these challenges, we propose the use of *partial models* (see e.g., Talvitie & Singh, 2008; Khetarpal et al., 2021; Zhao et al., 2021; Alver & Precup, 2023). Under this scenario, in its simplest implementation, the agent first detects the number of required partial models and then maintains a separate model for each relevant part of the state space, and in each update step before the local change, it performs updates to all of its models. Then, after the local change, it only updates the necessary models. And, in its scalable implementation, the agent emulates the idea of maintaining separate models by using separate heads and index lists. This conceptually simple idea naturally leads to a model, which is a collection of multiple partial models, that is both accurate across the state space and also quickly adaptable in the face of a local change in the environment.

We demonstrate the effectiveness of partial models by first instantiating them in the deep Dyna-Q agent and showing that they allow for achieving local adaptivity on the Local Change Adaptation (LoCA) setup (Van Seijen et al., 2020; Wan et al., 2022) of both the MountainCar and MiniGrid domains. We then test the generality of these results by instantiating partial models in modern deep MBRL agents like PlaNet (Hafner et al., 2019) and Dreamer (Hafner et al., 2020; 2021; 2023). Experiments on the LoCA setups of the pixel-based MuJoCo Reacher and RandomReacher

domains demonstrate that the use of partial models can indeed drastically improve the local adaptation capability of modern MBRL agents as well.

Key Contributions. The key contributions of this study are as follows: (i) Compared to the previous studies on local adaptation (Van Seijen et al., 2020; Wan et al., 2022; Rahimi-Kalahroudi et al., 2023), we propose two additional versions of the LoCA setup, with stochastic reward functions and non-stationary transition distributions, that are both more challenging than the original one (Sec. 3). (ii) We provide a detailed discussion on the challenges that arise in building locally adaptive deep MBRL agents and identify two additional challenges (Sec. 4). (iii) We propose two instantiations of the idea of partial models, a simple and scalable one, and demonstrate across four different domains that these types of models can allow for building locally adaptable deep MBRL agents (Sec. 5, 6 & 7).

2 Background

Reinforcement Learning. In RL (Sutton & Barto, 2018), an agent interacts with its environment through a sequence of actions to maximize its long-term cumulative reward. The interaction is usually modeled as a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, P, R, \rho_0, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are the (finite) set of state and actions, $P : \mathcal{S} \times \mathcal{A} \to \text{Dist}(\mathcal{S})$ is the transition distribution, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, $\rho_0 : \mathcal{S} \to \text{Dist}(\mathcal{S})$ is the initial state distribution, and $\gamma \in [0, 1)$ is the discount factor. The goal of the agent is to obtain a policy $\pi : \mathcal{S} \to \text{Dist}(\mathcal{A})$ that maximizes the expected sum of discounted rewards $E_\pi[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t, S_{t+1}) | S_0 \sim \rho_0]$.

Deep Model-Based Reinforcement Learning. In deep MBRL, an agent obtains a policy by planning with a learned model m of the environment which is represented with deep neural networks. Even though various deep MBRL agents have been proposed in the recent years (Moerland et al., 2023), notable state-of-the-art examples of them are PlaNet (Hafner et al., 2019) and Dreamer (Hafner et al., 2020; 2021; 2023), which perform decision-time and background planning, respectively (Sutton & Barto, 2018; Alver & Precup, 2022). These agents use deep neural networks in the implementation of their models and value functions, and store their experiences into a replay buffer. The stored experiences are then used for learning a model of the environment, which is then used for updating the value function of the agent.

The LoCA Setup. Inspired by studies on detecting model-based behavior in biological agents (Daw et al., 2011), Van Seijen et al. (2020) proposed the LoCA setup to evaluate model-based behavior in RL agents. Later, Wan et al. (2022) improved this setup by making it (i) simpler, (ii) less sensitive to hyperparameters and (iii) easily applicable to stochastic environments. The LoCA setup measures the adaptivity of an agent and thus serves as a preliminary yet important step towards the continual RL problem (Khetarpal et al., 2022).

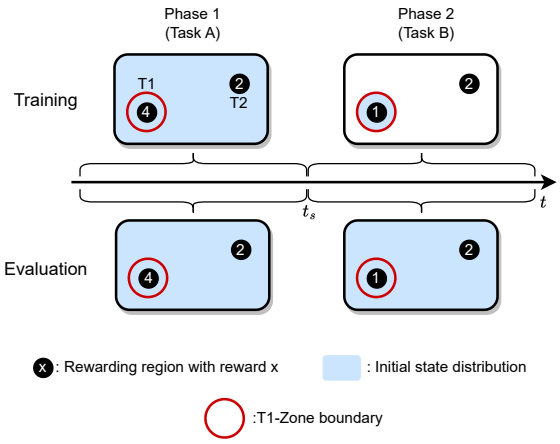


Figure 1: The LoCA setup of Wan et al. (2022). The values in the rewarding regions indicate the reward that is received in the corresponding region. t_s indicates the point in which the phase shift happens.

The LoCA setup considers an environment with two tasks, namely Task A and Task B, which differ only in their reward functions (see the first and second columns of Fig. 1, respectively). In each task, there are two rewarding regions, namely T1 and T2, and the reward function for the states outside of this region is always 0. For task A, the agent receives a reward of +4 upon entering T1 and +2 upon entering T2. For task B, however, entering T1 yields a reward of +1 (instead of +4 as in task A), and entering T2 yields the same reward as in task A, which is +2. Finally, the transition dynamics for a local area around T1 (called the T1-zone) is such that it is impossible to get out, once entered. Note that the difference between the two tasks is local and is only in the reward functions. Also note that while the difference is only local, the optimal policies are completely different: while the optimal policy for task A points to T1, the optimal policy for task B points to T2 (except when the agent is within the T1-zone, where the optimal policy again points to T1).

To test for adaptivity, the LoCA setup considers a scenario with two consecutive training phases, namely Phase 1 and Phase 2, which differ in the tasks and initial state distributions (see the first row of Fig. 1). Throughout Phase 1, the task is task A and the initial state is drawn uniformly from the entire state space. After Phase 1, Phase 2 begins and the task switches to task B (see point t_s in Fig. 1). Now, the initial state is drawn uniformly from only states within

Table 1: The details of the reward functions and transition distributions of LoCA, LoCA1 and LoCA2 setups.

	Reward Function		Transition Distribution	
	Phase 1	Phase 2	Phase 1	Phase 2
LoCA	Deterministic T1 region: +4 T2 region: +2 Rest: 0	Deterministic T1 region: +1 T2 region: +2 Rest: 0	Deterministic	Deterministic
LoCA1	Stochastic T1 region: $\mathcal{N}(+4, 0.5)$ T2 region: $\mathcal{N}(+2, 0.5)$ Rest: $\mathcal{N}(0, 0.5)$	Stochastic T1 region: $\mathcal{N}(+1, 0.5)$ T2 region: $\mathcal{N}(+2, 0.5)$ Rest: $\mathcal{N}(0, 0.5)$	Deterministic	Deterministic
LoCA2	Stochastic T1 region: $\mathcal{N}(+4, 0.5)$ T2 region: $\mathcal{N}(+2, 0.5)$ Rest: $\mathcal{N}(0, 0.5)$	Stochastic T1 region: $\mathcal{N}(+1, 0.5)$ T2 region: $\mathcal{N}(+2, 0.5)$ Rest: $\mathcal{N}(0, 0.5)$	Deterministic	Stochastic (Slippery T1-zone with slip probability 0.2)

the T1-zone; hence, the agent is stuck in the local area around T1 in Phase 2. Note that the agent does not receive any signal upon a phase switch. Throughout both of the training phases, the agent is periodically evaluated and the initial state for each evaluation episode is drawn uniformly over the entire state space (see the second row of Fig. 1). In each evaluation interval, the mean return is calculated and it is compared to the mean return of the corresponding optimal policy in that phase.

For the agent to reach optimal performance in Phase 2, it has to adapt to the new reward function that is present in task B: it has to change its policy from pointing towards T1 to pointing towards T2 for most of the states in the state space (states outside the T1-zone). Note that the agent has to perform this adaptation while only observing states within the T1-zone. Given a sufficient amount of time to train in both phases, the LoCA setup classifies an agent as adaptive if it is able to achieve close-to-optimal performance in both Phase 1 and Phase 2. If the agent only achieves near-optimal performance in Phase 1 but not in Phase 2, it is classified as non-adaptive. And lastly, the LoCA setup makes no assessment if the agent fails to reach close-to-optimal performance in Phase 1.

3 More Challenging Local Adaptation Setups

Even though the LoCA setup provides a way to evaluate model-based behavior in RL agents, it has two main shortcomings: it only considers (i) scenarios with deterministic reward functions, and (ii) scenarios with stationary transition distributions (see the first row of Table 1). In order to overcome these shortcomings, we propose two additional versions of the LoCA setup that are both more challenging that the original version: (i) LoCA1: a version of the setup with a stochastic reward function, and (ii) LoCA2: a version of the setup with both a stochastic reward function and a non-stationary transition distribution. More specifically, we convert the reward function into a stochastic one by turning it into a Gaussian: in Phase 1, upon entering T1 and T2 the agent receives a reward that is sampled from a Gaussian distribution with mean +4 and +2, respectively. And, we introduce non-stationarity into the transition dynamics by making the T1-zone slippery in Phase 2. More details on these setups can be found in the last two rows of Table 1.

4 Challenges in Building Adaptive Deep Model-Based Agents

Reason for Failing to Adapt. Previous studies on local adaptation (Van Seijen et al., 2020; Wan et al., 2022) have reported that popular deep MBRL agents, such as PlaNet (Hafner et al., 2019) and Dreamer (Hafner et al., 2020; 2021; 2023), fail in adapting to the local changes in their environments. The analysis of Wan et al. (2022) reveals that the key reason for the failure is because of the agent's inability in building and maintaining a sufficiently accurate model after a phase shift happens in the LoCA setup.

Challenge 1: Interference-Forgetting Dilemma. As pointed out in Wan et al. (2022) and Rahimi-Kalahroudi et al. (2023), one of the key challenges in building adaptive deep model-based agents is to overcome the *interference-forgetting dilemma*, which prevents an agents from building and maintaining a sufficiently accurate model after the phase change. In this dilemma, depending on the size of its replay buffer, the agent either faces the problem of interference or forgetting. More specifically, in the case of having a large replay buffer the agent faces the problem of *interference* which is caused by the interference of the stale data in the buffer with the new incoming data: e.g. in

the LoCA setup, after a phase shift happens the agent will be receiving +1 rewarding transitions, however, its replay buffer will still be containing the stale +4 rewarding transitions from Phase 1 and because of this the updates to the model corresponding to the T1 region will be interfered with the stale transitions. This will then result in an inaccurate model, which will then affect planning and thereby render the agent non-adaptive. And, in the case of using a small replay buffer the agent faces the problem of *forgetting* which is caused by the loss of the old-but-relevant data in the buffer: e.g. in the LoCA setup, after a phase shift happens the agent will be receiving +1 rewarding transitions and, due to the limited size of its buffer, it will over time lose the old-but-relevant +2 rewarding transitions from Phase 1 and because of this the model will over time fail in generating transitions with a reward of +2. This will again result in an inaccurate model, which will then again affect planning and thereby render the agent non-adaptive.

Challenge 2: Proper Model Update Challenge. Even if the agent manages to overcome the interference-forgetting dilemma, another very important challenge is in performing proper updates to the model so that it becomes and remains sufficiently accurate after a phase change. As modern deep MBRL agents update their models with randomly-sampled batches of transitions, they face the problem of ending up with biased models that are skewed towards the abundant transitions in the replay buffer: e.g. in the LoCA setup, after a phase shift happens the agent will be receiving lots of +1 rewarding transitions and because of this its model will be skewed towards generating transitions with a reward of +1. This problem will again result in an inaccurate model, which will then again affect planning and thereby render the agent non-adaptive. We refer to this challenge as the *proper model update challenge*.

Challenge 3: Quick Adaptation Challenge. Lastly, even though the previously introduced challenges are vital challenges in on their own, building agents that are not only adaptable but also *quickly* adaptable is another major challenge, as quick adaptation is another key characteristic of model-based behavior (Daw et al., 2011). We refer to this challenge as the *quick adaptation challenge*.

5 Partial Models for Building Adaptive Model-Based Agents

Before introducing the idea of partial models, we first introduce the terminology that we will use for the models and replay buffers of deep MBRL agents: following Van Hasselt et al. (2019), we refer to their models and replay buffers as *parametric* and *non-parametric* models , respectively.[1] We choose to use this terminology as it allows for a coherent presentation of the idea of partial models.

5.1 The Simplest Implementation

Details of the Architecture. In this study, we propose to use *partial models* as a way to address all of the three challenges presented in Sec. 4. In its simplest implementation, instead of having a single pair of non-parametric and parametric models, as is the default in deep MBRL agents, the agent maintains multiple pairs of non-parametric and parametric models that are each responsible for modeling different parts of the state space. An instantiation of this implementation with three partial models is illustrated in Fig. 2.

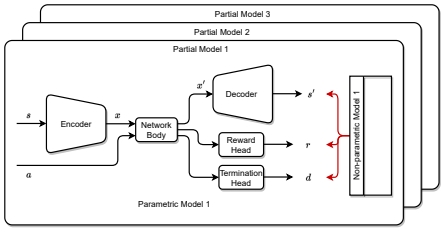


Figure 2: An instantiation of the simplest implementation of partial models with three pairs of non-parametric and parametric models. The red arrows indicate the direction of information flow from the non-parametric models to the parametric ones. The parametric model consist of an encoder e, network body nb, decoder d, reward head rh and termination head th.

Two important questions that arise in this context are (i) how many partial models to use, and (ii) how to determine which parts of the state space would each partial model be modeling. In the LoCA setup, as rewards allow for a natural way to cluster the state space, the agent uses of them to provide answers to these questions. More specifically, by using the data that is collected in the initial exploration phase, it first learns a neural embedding function E of the states such that states that have a similar reward are also closer in their embedding representation using contrastive learning (Hadsell et al., 2006; Dosovitskiy et al., 2014), and then runs a clustering algorithm, e.g. k-means clustering, over the embeddings and rewards to determine the number of partial models. Finally, it assigns each of the n models to parts of the state space that belongs to a different cluster.

In the learning of the neural embedding function, similar to Rahimi-Kalahroudi et al. (2023), we learn an embedding function $v = E(S)$, where E is a deep neural network that maps state S to an embedding vector v. This embedding

[1] While the former terminology is already obvious, the latter one also becomes obvious if one views the replay buffer as a model that provides accurate transitional data from the agent's past experiences.

function induces a distance metric in the state-space, such as $d(S_i, S_j) = ||E(S_i) - E(S_j)||_2$ for states S_i and S_j. We train the embedding function such that states that have similar rewards, have a smaller distance between the learned embeddings and states that have dissimilar rewards, have a relatively larger distance between them in the embedding space. Let S' represent a state that is in the reward proximity of state S, i.e. $|r(S) - r(S')| < p$ for some p, and $\bar{\mathbf{S}}$ represent a set of states that are not, and let $D = \{S, S', \bar{\mathbf{S}}\}$ be a dataset of their collection. We train the embedding function to minimize the following loss function:

$$L(D) = \sum_{(S,S',\bar{\mathbf{S}})\in D} ||E(S) - E(S')||_2^2 + \left(\beta - \sum_{\bar{S}\in\bar{\mathbf{S}}} ||E(S) - E(\bar{S})||_2^2\right)^2,$$

where $\beta > 0$ is a hyperparameter. This loss function trains the embedding of states S and S' to be closer by minimizing their distance towards zero, while pushing the embedding of states $\bar{\mathbf{S}}$ to be on average farther, with a cumulative squared distance value close to β.

Details of Achieving Adaptivity. Once these issues are addressed, the agent would be able to use partial models to achieve local adaptivity. Under this scenario, in Phase 1, while performing updates to its parametric models $\mathcal{P} = \{m_p^i = (e, nb, d, rh, th)\}_{i=1}^n$ and value function Q, the agent updates (i) all of its n parametric models with their corresponding non-parametric models $\mathcal{NP} = \{m_{np}^i\}_{i=1}^n$ and (ii) its value function with all of its parametric models in $\mathcal{P}$. And, in Phase 2, it first clears the stale samples in its corresponding non-parametric model and then updates (i) only the necessary parametric model with its corresponding non-parametric one and (ii) its value function with all of its parametric models.

Note that in order to only perform the necessary updates in Phase 2, the agent would have to first (i) detect the phase change and then (ii) infer which of its parametric models requires an update, as the LoCA setup does not provide this information. To overcome these issues, the agent trains a classifier C to distinguish between the rewards it has seen so far and the anomalous ones, and after detecting the change, it identifies the model that requires an update by first passing the state corresponding to the anomalous reward through the embedding function and then identifying which cluster it belongs to. The main pseudocode of how to achieve adaptivity with the simple implementation of partial models is presented in Alg. 1. For more details on the ICOC algorithm, we refer the reader to Alg. 2 in App. A.

How does this implementation address the challenges? Having modularity in the non-parametric model structure allows the agent to mitigate the interference-forgetting dilemma as the agent now clears the stale samples in its corresponding non-parametric model upon a detection of a phase change, and it also stores the old-but-relevant data in a separate non-parametric model. This modularity also addresses the proper model update challenge as the agent now samples an equal amount of transitions from

Algorithm 1 Pseudocode for achieving adaptivity with the Simple and Scalable Implementation of Partial Models. The red and blue colored parts are specific to the simple and scalable implementations, respectively (i.e. the simple implementation does not contain the blue parts and the scalable implementation does not contain the red parts).

$\mathcal{D} \leftarrow$ gather m samples via a random policy
$\mathcal{CC}, C, E, n \leftarrow$ IDENTIFYCLUSTERS&OBTAINCLASSIFIER($\mathcal{D}$) (ICOC; $\mathcal{CC}$ is a dictionary of cluster centers, C is the classifier, E is the neural embedding function and n is the number of clusters)
$\mathcal{NP} \leftarrow \{m_{np}^i = [\,]\}_{i=1}^n$ (initialize n empty non-parametric models)
$\mathcal{P} \leftarrow \{m_p^i = (e, nb, d, rh, th)\}_{i=1}^n$ (initialize n parametric models)
$m_{np} \leftarrow [\,]$ (initialize an empty non-parametric model)
$\mathcal{IL} \leftarrow \{\mathcal{I}^i = [\,]\}_{i=1}^n$ (initialize n empty index lists)
$m_p \leftarrow (e, nb, d, \{rh^i\}_{i=1}^n, th)$ (initialize a parametric model with n rh's)
Initialize the parameters of the parametric models and of Q

$phase \leftarrow 1$ (the current LoCA phase)
while training continues **do**
    $S \leftarrow$ reset environment
    while not done **do**
        R, S', done $\leftarrow$ environment(ϵ-greedy($Q(S)$))
        if C(concatenate($E(S), R$) = "anomalous" **then**
            $phase \leftarrow 2$
            $k_{\text{req\_upd}} \leftarrow$ detect which model requires update by identifying the cluster ID that $E(S)$ is closest using $\mathcal{CC}$
            Clear $\mathcal{NP}[k_{\text{req\_upd}}]$ / $\mathcal{IL}[k_{\text{req\_upd}}]$
            $\mathcal{D} \leftarrow$ gather m samples via a random policy
            $\mathcal{CC}_{\text{new}}, C_{\text{new}}, E, \_ \leftarrow$ ICOC($\mathcal{D} + \mathcal{NP}$ / $\mathcal{D}$ + m_{np} + $\mathcal{IL}$)
            $\mathcal{CC}, C \leftarrow$ compare $\mathcal{CC}$ and $\mathcal{CC}_{\text{new}}$, identify similar clusters using the cluster centers, and retain the old cluster IDs of these similar ones in $\mathcal{CC}_{\text{new}}$ and C_{new}
        $k \leftarrow$ identify the model class by C(concatenate($E(S), R$)
        $\mathcal{NP}[k]$ / $m_{np} \leftarrow \mathcal{NP}[k]$ / m_{np} +{$(S, A, R, S'$, done)}
        $\mathcal{IL}[k] \leftarrow \mathcal{IL}[k]$ + {current time step t}
        if $phase = 1$ **then**
            for i in $\{1...n\}$ **do**
                $\mathcal{B} \leftarrow$ sample_batch($\mathcal{NP}[i]/m_{np}, \mathcal{IL}[i]$)
                Update $\mathcal{P}[i]$ / $m_p[i]$ with $\mathcal{B}$
                Update Q with the predictions of $\mathcal{P}[i]$ / $m_p[i]$ on $\mathcal{B}$
        if $phase = 2$ **then**
            $\mathcal{B} \leftarrow$ sample_batch($\mathcal{NP}[k_{\text{req\_upd}}]/m_{np}, \mathcal{IL}[k_{\text{req\_upd}}]$)
            Update $\mathcal{P}[k_{\text{req\_upd}}]$ with $\mathcal{B}$
            Freeze (e, nb, d, th) in m_p and only update $rh^{k_{\text{req\_upd}}}$ with $\mathcal{B}$
            for i in $\{1...n\}$ **do**
                $\mathcal{B} \leftarrow$ sample_batch($\mathcal{NP}[i]/m_{np}, \mathcal{IL}[i]$)
                Update Q with the predictions of $\mathcal{P}[i]$ / $m_p[i]$ on $\mathcal{B}$
        $S \leftarrow S'$

each of its non-parametric models to update its corresponding parametric ones. Lastly, the modularity in the parametric model structure also addresses the quick adaptation challenge as the agent now infers which of its parametric models requires an update upon a phase change and it focuses on updating the one that requires it.

5.2 A Scalable Implementation

Even though the simple implementation allows for building adaptive MBRL agents, maintaining multiple models is not a scalable approach as the memory and computation requirements grow linearly with each model. Thus, we now present a scalable implementation of the idea of partial models that again retains simplicity.

Details of the Architecture. In this implementation, the agent now maintains a single non-parametric and parametric model that emulates the idea of having multiple models: after determining how many partial models to use and which parts of the state space would each partial model be modeling through the same procedure as in Sec. 5.1, the non-parametric model m_{np} now maintains a separate index list $\mathcal{IL} = \{\mathcal{I}^i\}_{i=1}^n$ for each of the partial models, and the parametric model $m_p = (e, nb, d, \{rh^i\}_{i=1}^n, th)$ now consists of multiple reward heads each corresponding to a different partial model. An instantiation of this implementation with three partial models is illustrated in Fig. 3.

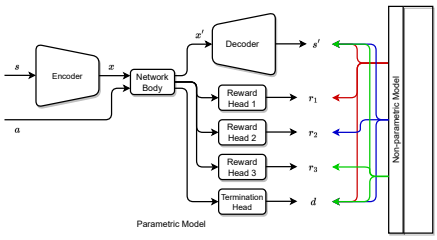


Figure 3: An instantiation of a scalable implementation of partial models with a non-parametric model consisting of three index lists and a parametric one consisting of three reward heads. The red, blue and green arrows indicate the direction of information flow from the non-parametric model to the parametric one.

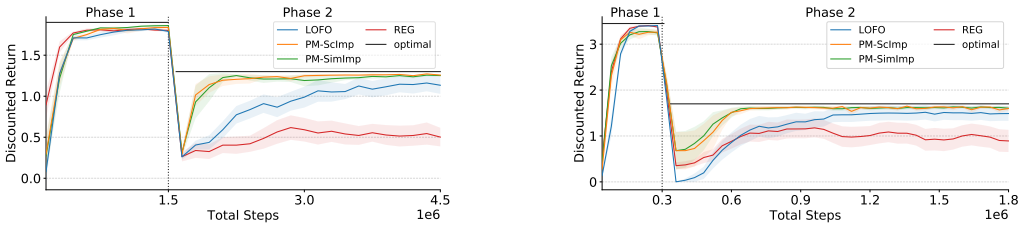
Details of Achieving Adaptivity. Under this scenario, in Phase 1, while performing updates to its parametric model m_p, the agent (i) first samples an equal amount of transitions by using each index list $\mathcal{I}^i$ in its non-parametric model m_{np} and (ii) then it updates its encoder e, network body nb, decoder d and termination head th with all of these transitions. However, while updating its reward heads rh^i, it only updates them with the transitions that are sampled from their corresponding index lists. And, in Phase 2, the agent (i) first clears the list that corresponds to the indices of the stale samples, and (ii) then freezes all the parameters of its parametric model except for the parameters of the reward head that requires adaptation. Then, it updates only these parameters with the transitions that are sampled according to the corresponding index list. Note that, in both of the phases, while performing updates to its value function Q, the agent makes use of the transitions that are generated from all of the heads of its parametric model. Finally, (i) the detection of the phase change and (ii) the inference of which reward head to perform updates to is done in a similar fashion as in Sec. 5.1. The main pseudocode of how to achieve adaptivity with the scalable implementation of partial models is presented in Alg. 1.

How does this implementation address the challenges? Note that this implementation also addresses both the interference-forgetting dilemma and the proper model update challenge as the agent again maintains modularity in the non-parametric model structure through the separate index lists. And, it also addresses the quick adaptation challenge as the agent again maintains modularity in its parametric model through its separate reward heads.

6 Deep Dyna-Q Experiments

Following previous studies on local adaptation (Wan et al., 2022; Rahimi-Kalahroudi et al., 2023), we start by performing experiments with the deep Dyna-Q agent as it is a simple MBRL agent that is reflective of many of the properties of its state-of-the-art counterparts (Hafner et al., 2019; 2020; 2021; 2023). The details of all of the experiments in this section can be found in App. B & C.

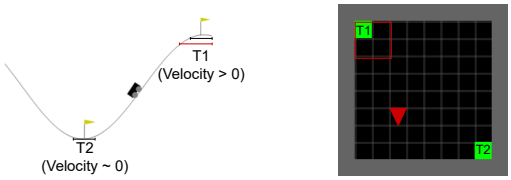
Environmental Details. We perform our evaluation on the LoCA setup of two domains: (i) the MountainCar domain (MountainCarLoCA), and (ii) a simple MiniGrid domain (MiniGridLoCA) (Chevalier-Boisvert et al., 2018). We choose the domains as they were also used in the experiments of previous studies regarding the LoCA setup (Van Seijen et al., 2020; Wan et al., 2022; Rahimi-Kalahroudi et al., 2023). The MountainCarLoCA setup (Fig. 4a) is built on top of the classical MountainCar domain in which an under-powered cart has to move to certain locations by taking its position and velocity information from the environment as an input. In this setup, there are two rewarding regions that act as terminal states: (i) T1 which requires the agent to be at the top of the hill and have a velocity greater than zero, and (ii) T2 which requires the agent to be at the bottom of the hill and have a velocity close to zero. The MiniGridLoCA setup (Fig. 4b) is built on top of a simple MiniGrid domain in which the agent has to navigate to the green goal cells by taking a top-down image of the environment as input. In this setup, there are again two rewarding regions that act as terminal states: (i) T1 which is at the top-left corner, and (ii) T2 which is at the bottom-right corner.



(a) MountainCarLoCA (b) MiniGridLoCA

Figure 5: Plots showing the learning curves of the deep Dyna-Q agents that are referred to as PM-SimImp, PM-ScImp, REG and LOFO on the (a) MountainCarLoCA and (b) MiniGridLoCA setups. Each learning curve is an average discounted return over 20 runs and the shaded area represents the confidence intervals. The maximum possible return in each phase is represented by a solid black line.

Deep Dyna-Q with Partial Models and Baselines. In order to demonstrate the flexibility in implementing partial models, we implement two deep Dyna-Q agents with varying partial model implementations: (i) in the first one the agent employs a simple implementation of partial models (PM-SimImp), and (ii) in the second one the agent employs a scalable implementation of partial models (PM-ScImp). We compare these agents with two baseline agents from Rahimi-Kalahroudi et al. (2023): (i) a regular deep Dyna-Q agent (REG), and (ii) a deep Dyna-Q agent with a specifically designed replay buffer that removes the oldest sample from a local neighbourhood of a new sample in its replay buffer to address the interference-forgetting dilemma (LOFO). For both of these baselines, we choose the best performing agents from Rahimi-Kalahroudi et al. (2023).



(a) MountainCarLoCA (b) MiniGridLoCA

Figure 4: Illustration of the MountainCarLoCA and MiniGridLoCA setups. The solid red lines indicate the T1-zone boundaries in Phase 2 of the LoCA setup.

Quantitative Analysis. Fig. 5a & 5b show the learning curves of the different deep Dyna-Q agents on the MountainCarLoCA and MiniGridLoCA setups, respectively. As can be seen, while all the agents reach close to optimal performance in Phase 1, the REG agent fails in adapting to the local change in Phase 2 as it suffers from the interference-forgetting dilemma. And, even though the PM-SimImp, PM-ScImp and LOFO agents are all able to display adaptability in Phase 2, which makes all of them adaptive MBRL agents as per the LoCA setup, the PM-SimImp and PM-ScImp agents are able adapt much faster compared to the LOFO agent, demonstrating that besides addressing the interference-forgetting dilemma and the proper model update challenge, they also address the quick adaptation challenge. In order to demonstrate the generality of the performance of partial models across different LoCA setups, we also evaluated the different deep Dyna-Q agents on the LoCA1 and LoCA2 setups of the MountainCar and MiniGrid domains. Results in Fig. 10 & 11 show that a similar performance trend also holds in these setups.

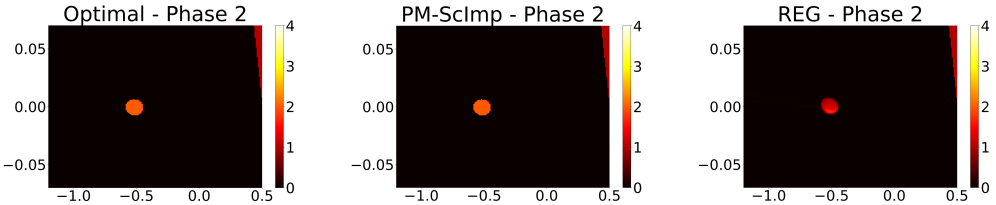


Figure 6: Visualization of the predicted rewards of the parametric models of the optimal, PM-ScImp and REG agents at the end of Phase 2 in the MountainCarLoCA setup. In each heatmap, the x axis represents the agent's position, and the y axis represents its velocity.

Qualitative Analysis I. To have a better understanding of how the use of partial models mitigates the interference-forgetting dilemma and the proper model update challenge, we look at the reward predictions of the PM-ScImp and REG agents at the end of Phase 2 and compare them with an optimal agent in the MountainCarLoCA setup. The

visualizations are shown in Fig. 6. As can be seen, the PM-ScImp agent is able to correctly adapt its reward function to around +1 for the T1 terminal states (triangular region at the top-right corner) and maintain its reward function around +2 for the T2 terminal states (circular region in the middle), almost matching the performance of the optimal agent. On the other hand, as it is not even able to address the interference-forgetting dilemma, the REG agent fails in correctly predicting the reward function for the T2 terminal states.

Qualitative Analysis II. To see if the PM-SimImp and PM-ScImp agents work as intended, we also look at the number of partial models they use and the assignment places of these models in the state space. With every run, we consistently observe that these agents make use of three partial models: (i) one that is assigned to the T1 rewarding region, (ii) one that is assigned to the T2 rewarding region, and (iii) one that is assigned to the region outside of the T1 and T2 regions. For example, the reward predictions of the PM-ScImp agent (see Fig. 6) were generated by three different partial models: (i) the triangular region at the top-right corner was generated with the partial model that is responsible for the T1 terminal states, (ii) the circular region in the middle was generated with a partial model that is responsible for the T2 terminal states, and (iii) the rest was generated with a partial model that is responsible for the states outside of the T1 and T2 terminal states.

Discussion on the Memory and Computation Costs. In our experiments, we observed that while the experiments with PM-SimImp agent took thrice as much memory and wall clock time than the experiments with the REG agent, the experiments with PM-ScImp agent took around the same memory and wall clock time of the experiments with the REG agent. This justifies that the implementation proposed in Sec. 5.2 is indeed a scalable implementation.

7 PlaNet and Dreamer Experiments

To demonstrate the generality of the use of partial models in building adaptive deep model-based agents, we also employed partial models in modern deep MBRL agents such as PlaNet (Hafner et al., 2019) and Dreamer (Hafner et al., 2020; 2021; 2023) and evaluated their performance in the LoCA setup. The details of all of the experiments in this section can be found in App D & E.

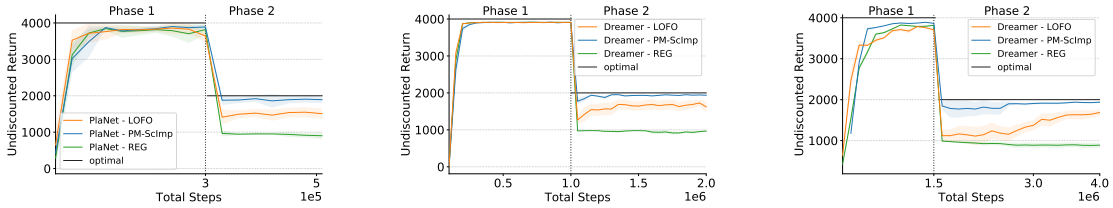
Environmental Details. We evaluate these agents on the LoCA setup of two domains: (i) the pixel-based Reacher domain that was introduced by Wan et al. (2022) (ReacherLoCA), and (ii) the randomized version of the pixel-based Reacher domain (RandomReacherLoCA). Again, we choose these domains as they were used in the experiments of previous studies regarding the LoCA setup (Wan et al., 2022; Rahimi-Kalahroudi et al., 2023). The ReacherLoCA setup (Fig. 7a) is built on top of the continuous-action Reacher domain (Tassa et al., 2018) in which the agent has to move the tip of the outer bar to the target positions and keep it there till the end of the episode (which takes 1000 time steps) by taking an 64x64 top-down image of the environment as input. In this variation, there are two rewarding regions: (i) T1 which is at the top-right quadrant, and (ii) T2 which is at the bottom-left quadrant. The RandomReacherLoCA setup (Fig. 7b) is a more complicated extension of the ReacherLoCA setup where the locations of the two terminal states keep randomly changing from one episode to another, while still being exactly opposite to each other. In this setup, the T2 terminal state is colored differently from the T1 one so that the agent can differentiate between them.

(a) ReacherLoCA (b) RandomReacherLoCA

Figure 7: Illustration of the ReacherLoCA and RandomReacherLoCA setups. The solid red lines indicate the T1-zone boundaries in Phase 2 of the LoCA setup.

PlaNet/Dreamer with Partial Models and Baselines. As we have already demonstrated in Sec. 6 that the two different implementations of partial models both allow for building adaptive deep MBRL agents, in this section, we only run experiments with the PlaNet and Dreamer agents that employ a scalable implementation of partial models (PlaNet - PM-ScImp, Dreamer - PM-ScImp). We again compare these agents with the two baseline agents from Rahimi-Kalahroudi et al. (2023): (i) the regular PlaNet and Dreamer agents (PlaNet - REG, Dreamer - REG), and (ii) the PlaNet and Dreamer agents with LOFO (PlaNet - LOFO, Dreamer - LOFO). Again, for both of these baselines, we choose the best performing agents from Rahimi-Kalahroudi et al. (2023).

Quantitative Analysis. Fig. 8a & 8b show the learning curves of the different PlaNet and Dreamer agents on the ReacherLoCA setup, and Fig. 8c shows the learning curves of the Dreamer agents on the RandomReacherLoCA setup. We observe again that in both of the setups, while all the agents reach close to optimal performance in Phase 1, the REG agents fail in adapting to the local change in Phase 2 as they again suffer from the interference-forgetting



(a) ReacherLoCA (b) ReacherLoCA (c) RandomReacherLoCA

Figure 8: Plots showing the learning curves of the PlaNet - PM-ScImp, Dreamer - PM-ScImp, PlaNet - REG, Dreamer - REG, PlaNet - LOFO, Dreamer - LOFO agents on the (a, b) ReacherLoCA and (c) RandomReacherLoCA setups. Each learning curve is an average undiscounted return over 10 runs and the shaded area represents the confidence intervals. The maximum possible return in each phase is represented by a solid black line.

dilemma. And, again, even though both the PM-ScImp and LOFO agents are able to display adaptability in Phase 2, which makes both of them adaptive MBRL agents as per the LoCA setup, the PM-ScImp agents are able to adapt much faster compared to the LOFO agents, again demonstrating that besides addressing the two challenges, they also address the quick adaptation challenge. In order to demonstrate the generality of the performance of partial models across different LoCA setups, we also evaluated the these agents on the LoCA1 and LoCA2 setups of the Reacher and RandomReacher domains. Results in Fig. 12 & 13 show that a similar performance trend also holds in these setups.

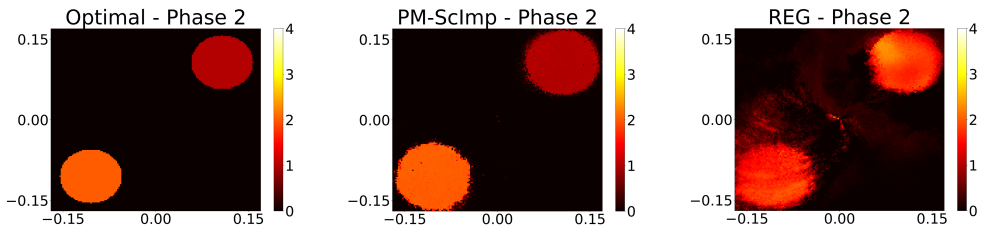


Figure 9: Visualization of the predicted rewards of the parametric models of the optimal, Dreamer - PM-ScImp and Dreamer - REG agents at the end of Phase 2 in the ReacherLoCA setup. Each point on the heatmaps represents the agent's position in the Reacher environment.

Qualitative Analysis I. Again, to have a better understanding of how the use of partial models mitigates the interference-forgetting dilemma and the proper model update challenge, we look at the reward predictions of the Dreamer - PM-ScImp and Dreamer - REG agents at the end of Phase 2 and compare them with an optimal agent in the ReacherLoCA setup. The visualizations in Fig. 9 show that the PM-ScImp agent is able to correctly adapt its reward function to around +1 for the T1 rewarding region (circular region at the top-right quadrant) and maintain its reward function around +2 for the T2 rewarding regions (circular region at the bottom-left quadrant), again almost matching the optimal agent. On the other hand, as it is again not even able to address the interference-forgetting dilemma, the REG agent fails in correctly predicting the reward function for both the T1 and T2 rewarding regions.

Qualitative Analysis II. Again, to see if the PM-ScImp agents works as intended, we also look at the number of partial models they use and the assignment places of these models in the state space. Similar to our results in Sec. 6, with every run, we consistently observe that these agents make use of three partial models that are assigned to the T1 rewarding region, the T2 rewarding region and the region outside of the T1 and T2 regions. For example, the reward predictions of the Dreamer - PM-ScImp agent (see Fig. 9) were generated by three different partial models: (i) the circular region at the top-right quadrant was generated with the partial model that is responsible for the T1 terminal states, (ii) the circular region at the bottom-left quadrant was generated with a partial model that is responsible for the T2 terminal states, and (iii) the rest was generated with a partial model that is responsible for the states outside of the T1 and T2 terminal states.

Discussion on the Memory and Computation Costs. In our experiments, we observed again that the experiments with PM-ScImp agent took around the same memory and wall clock time of the experiments with the REG agent, which again justifies that the scalable implementation of partial models (Sec. 5.2) is indeed scalable in terms of the memory and computation requirements.

8 Related Work

Adaptability in Model-Based RL. Our study focuses on building adaptive deep MBRL agents in the LoCA setup, therefore it is mostly related to the studies of Van Seijen et al. (2020) and Wan et al. (2022). As mentioned previously, while the former of these studies introduced the setup, the latter one improved it in several ways (see Sec. 2). Our study is also related to the study of Rahimi-Kalahroudi et al. (2023) which proposed the use of a special kind of replay buffer, called LOFO, for achieving adaptivity in the LoCA setup. This replay buffer overcomes the interference-forgetting dilemma by first detecting the oldest sample from a local neighbourhood of a new sample and then by removing it from the replay buffer. However, in this study, rather than focusing on a solution on the replay buffer side, we focused on a one that exploits the power of having modularity in the model structure. We also note that, unlike Rahimi-Kalahroudi et al. (2023), our solution does not rely on performing a search through the whole replay buffer at every time step, which brings computational complexity benefits to our approach over LOFO.

Partial Models in RL. In the context of RL, partial models are models that are over certain aspects of the environment. In the literature, various forms of partial models have been considered: e.g. (i) Talvitie & Singh (2008) considers models that are partial at the pixel level, i.e. the learned model only models certain pixels in the input observation, (ii) Khetarpal et al. (2021) considers models that are partial at the action level, i.e. the learned model only models the consequences of certain actions, and (iii) Zhao et al. (2021) and Alver & Precup (2023) consider models that are partial at the feature level, i.e. the learned model only models the evolution of certain features. Inspired by these studies, our study considers models that are partial at the state level, i.e. the learned models only model the transitions and reward functions in certain regions of the state space. However, unlike prior studies, in our study the MBRL agent maintains multiple partial models and effectively uses them for achieving local adaptivity. It is also important to note that, in this study, the use of partial models is explicitly for the purposes of adaptivity, and not for gaining any benefits in the computational complexity of the planning process like in Khetarpal et al. (2021) and Alver & Precup (2023).

Ensemble Learning in RL. As our approach makes use of multiple partial models, it can be considered as an ensemble learning method. However, while there has been several studies that make use of ensemble learning methods in RL (Song et al., 2023), our approach differs in that we apply ensemble learning techniques to model-based RL algorithms.

The Continual RL Problem. The LoCA setup measures the adaptivity of an agent and thus serves as a preliminary yet important step towards the ambitious continual RL problem (Parisi et al., 2019; Khetarpal et al., 2022; Kessler et al., 2022). It specifies a particular kind of continual RL problem which involves a local environmental change. However, it should also be noted that, unlike regular continual RL problems where the agent has to prevent catastrophic forgetting and perform well across all the tasks that it has seen so far, in the LoCA setup the only thing that is of importance is the agent's performance in the current task.

The Transfer Learnıng Problem. As there is a need to solve multiple tasks, the LoCA setup is also related to the well-known transfer learning problem (Lazaric, 2012; Taylor & Stone, 2009; Zhu et al., 2023). However, unlike the transfer learning problem, in the LoCA setup the agent is not informed of which task it has to solve.

Model-Based RL for Contınual RL. While there has been several studies that focus on developing MBRL algorithms for the continual learning and transfer learning problems (Zhang et al., 2019; Huang et al., 2021; Nguyen et al., 2012; Boloka et al., 2021), none of these algorithms directly address the challenges that were depicted in Sec. 4. Therefore, generally speaking, they are not likely to demonstrate adaptivity in the LoCA setup.

9 Conclusion and Discussion

In this study, we focused on one of the key characteristics of model-based behavior: the ability to adapt to local changes in the environment. After discussing the three main challenges for performing adaptive deep MBRL, we proposed the use of partial models as a way to mitigate these challenges. We provided two specific implementations, a simple and a scalable one, and demonstrated through experiments that when employed in state-of-the-art deep model-based agents, such as PlaNet and Dreamer, these models do not only allow for adaptivity to the local changes but also allow for quick adaptation to such changes. We believe that this is an important step towards the ambitious goal of building continual RL agents as continual RL scenarios often require adaptation to changes in the environment.

It is important to note that even though we have only considered specific local adaptation setups (see Sec. 3), the idea of having *modularity in the model* of the agent is a general idea that can also be leveraged in other (local or non-local) adaptation setups. Finally, we note that, similar to previous studies in the literature (Rahimi-Kalahroudi et al., 2023), our study also assumes (i) a good initial exploration phase and (ii) a good way to measure similarity between states, which we do with their corresponding rewards, so that it is possible to learn neural embeddings of them.

ACKNOWLEDGMENTS

This project has been partly funded by an NSERC Discovery grant and the Canada-CIFAR AI Chair program. We would like to thank the anonymous reviewers for providing critical and constructive feedback.

REFERENCES

Safa Alver and Doina Precup. Understanding decision-time vs. background planning in model-based reinforcement learning. *arXiv preprint arXiv:2206.08442*, 2022.

Safa Alver and Doina Precup. Minimal value-equivalent partial models for scalable and robust planning in lifelong reinforcement learning. In Sarath Chandar, Razvan Pascanu, Hanie Sedghi, and Doina Precup (eds.), *Proceedings of The 2nd Conference on Lifelong Learning Agents*, volume 232 of *Proceedings of Machine Learning Research*, pp. 548–567. PMLR, 22–25 Aug 2023. URL https://proceedings.mlr.press/v232/alver23a.html.

Tlou Boloka, Ndivhuwo Makondo, and Benjamin Rosman. Knowledge transfer using model-based deep reinforcement learning. In *2021 Southern African Universities Power Engineering Conference/Robotics and Mechatronics/Pattern Recognition Association of South Africa (SAUPEC/RobMech/PRASA)*, pp. 1–6. IEEE, 2021.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for gymnasium, 2018. URL https://github.com/Farama-Foundation/Minigrid.

Nathaniel D Daw, Samuel J Gershman, Ben Seymour, Peter Dayan, and Raymond J Dolan. Model-based influences on humans' choices and striatal prediction errors. *Neuron*, 69(6):1204–1215, 2011.

Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. *Advances in neural information processing systems*, 27, 2014.

Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 2, pp. 1735–1742. IEEE, 2006.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pp. 2555–2565. PMLR, 2019.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0oabwyZbOu.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Yizhou Huang, Kevin Xie, Homanga Bharadhwaj, and Florian Shkurti. Continual model-based reinforcement learning with hypernetworks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 799–805. IEEE, 2021.

Samuel Kessler, Jack Parker-Holder, Philip Ball, Stefan Zohren, and Stephen J Roberts. Same state, different task: Continual reinforcement learning without interference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 7143–7151, 2022.

Khimya Khetarpal, Zafarali Ahmed, Gheorghe Comanici, and Doina Precup. Temporally abstract partial models. *Advances in Neural Information Processing Systems*, 34:1979–1991, 2021.

Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, 2022.

Alessandro Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning: State-of-the-Art*, pp. 143–173. Springer, 2012.

Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

Trung Nguyen, Tomi Silander, and Tze Leong. Transferring expectations in model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 25, 2012.

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019.

Ali Rahimi-Kalahroudi, Janarthanan Rajendran, Ida Momennejad, Harm van Seijen, and Sarath Chandar. Replay buffer with local forgetting for adapting to local environment changes in deep model-based reinforcement learning. In Sarath Chandar, Razvan Pascanu, Hanie Sedghi, and Doina Precup (eds.), *Proceedings of The 2nd Conference on Lifelong Learning Agents*, volume 232 of *Proceedings of Machine Learning Research*, pp. 21–42. PMLR, 22–25 Aug 2023. URL https://proceedings.mlr.press/v232/rahimi-kalahroudi23a.html.

Yanjie Song, Ponnuthurai Nagaratnam Suganthan, Witold Pedrycz, Junwei Ou, Yongming He, Yingwu Chen, and Yutong Wu. Ensemble reinforcement learning: A survey. *Applied Soft Computing*, pp. 110975, 2023.

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

Erik Talvitie and Satinder Singh. Simple local models for complex dynamical systems. *Advances in Neural Information Processing Systems*, 21, 2008.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

Hado P Van Hasselt, Matteo Hessel, and John Aslanides. When to use parametric models in reinforcement learning? *Advances in Neural Information Processing Systems*, 32, 2019.

Harm Van Seijen, Hadi Nekoei, Evan Racah, and Sarath Chandar. The loca regret: A consistent metric to evaluate model-based behavior in reinforcement learning. *Advances in Neural Information Processing Systems*, 33:6562–6572, 2020.

Yi Wan, Ali Rahimi-Kalahroudi, Janarthanan Rajendran, Ida Momennejad, Sarath Chandar, and Harm H Van Seijen. Towards evaluating adaptivity of model-based reinforcement learning methods. In *International Conference on Machine Learning*, pp. 22536–22561. PMLR, 2022.

Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. Solar: Deep structured representations for model-based reinforcement learning. In *International conference on machine learning*, pp. 7444–7453. PMLR, 2019.

Mingde Zhao, Zhen Liu, Sitao Luan, Shuyuan Zhang, Doina Precup, and Yoshua Bengio. A consciousness-inspired planning agent for model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Zhuangdi Zhu, Kaixiang Lin, Anil K Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

A Pseudocode of the ICOC Algorithm

The detailed pseudocode of the ICOC algorithm for identifying the clusters and obtaining a classifier is depicted in Alg. 2. Note that the details of how this pseudocode is actually implemented can change from one agent to the other (e.g. Deep Dyna-Q, Dreamer, etc.), however, the main structure will remain the same.

Algorithm 2 Detailed pseudocode for identifying the clusters and obtaining a classifier.

procedure IdentifyClusters&ObtainAClassifier($\mathcal{D}$)
  $E \leftarrow$ learn with $\mathcal{D}$ a neural embedding function of states such that states that have a similar reward are also closer in their neural embedding representation (e.g. by using the contrastive learning method used in Rahimi-Kalahroudi et al. (2023))
  $\mathcal{D}_u \leftarrow$ form an unlabelled dataset of vectors by concatenating the neural embedding of the state $E(S)$ and the reward R of each transition in $\mathcal{D}$
  $n \leftarrow$ identify the optimal number of clusters by running a clustering algorithm over the neural embeddings part of $\mathcal{D}_u$ (e.g. by using k-means clustering and the elbow method)
  $\mathcal{CC} \leftarrow$ create a dictionary where the keys are the cluster IDs and the values are the corresponding neural embedding and reward components of the cluster centers ($\mathcal{CC}$ comes from cluster centers)
  $\mathcal{D}_l \leftarrow$ form a labeled dataset of $\mathcal{D}_u$ by using their cluster IDs (i.e. labels are the cluster IDs)
  $\mathcal{D}_a \leftarrow$ create an artificial version of $\mathcal{D}_u$ in which the reward components of the vectors are replaced with artificial rewards that are not seen during training and label these vectors as "anomalous"
  $\mathcal{D}_l \leftarrow \mathcal{D}_l + \mathcal{D}_a$ (unify the two datasets into one)
  $C \leftarrow$ train a classifier over $\mathcal{D}_l$
  return $\mathcal{CC}$, C, E, n

B Details of the MountainCarLoCA Experiments

B.1 Experimental Setup

In the MountainCarLoCA setup (Van Seijen et al., 2020), T1 is located at the top of the mountain (position > 0.5, and velocity > 0), and T2 is located at the valley ($(\text{position}+0.52)^2+100\times\text{velocity}^2 \leq 0.07^2$). The boundary for the initial state distribution encircles all the states within $0.4 \leq$ position ≤ 0.5 and $0 \leq$ velocity ≤ 0.07. The discount factor is $\lambda = 0.99$. For each evaluation, the agent is initialized roughly in the middle of T1 and T2 ($-0.2 \leq$ position ≤ -0.1 and $-0.01 \leq$ velocity ≤ 0.01). Table 2 shows the details of the experimental setup that was used to evaluate the adaptivity of the deep Dyna-Q agents in MountainCarLoCA.

B.2 Hyperparameters

In our experiments, we used the same deep Dyna-Q agent that was used by Wan et al. (2022) (we refer to it as REG in this study). However, similar to Rahimi-Kalahroudi et al. (2023), instead of having separate neural networks for different actions in each part of the model (transition, reward, and termination models), we use just one network and concatenate the action with the output of the middle layer (the one that has 63 output units) and then feed it to the next layer. Table 3 summarizes the important hyperparameters of the REG agent. Note that these are the same hyperparameters that were used by Rahimi-Kalahroudi et al. (2023). For the PM-SimImp and PM-ScImp agents we have just extended the REG agent in a way that is described in Sec. 5 and thus the hyperparameters are the same as the REG agent.

As the LOFO agent is also built on top of REG agent, the base hyperparameters of it is again the same as the REG agent and the additional hyperparameters of it can be reached at Table 4. Note again that these are the same hyperparameters that were used by Rahimi-Kalahroudi et al. (2023). For our neural embedding function, we have used the same hyperparameters with the LOFO agent.

C Details of the MiniGridLoCA Experiments

C.1 Experimental Setup

The MiniGridLoCA setup was implemented on top of the MiniGrid suite (Chevalier-Boisvert et al., 2018). Table 5 shows the details of the experimental setup that was used to evaluate the adaptivity of the deep Dyna-Q agents in MiniGridLoCA.

Table 2: Experimental setup for the deep Dyna-Q agent in MountainCarLoCA.

Initial state distributions	Phase 1 training	Uniform distribution over the entire state space
	Phase 1 evaluation	Uniform distribution over a small region
	Phase 2 training	Uniform distribution over states within the red boundary
	Phase 2 evaluation	Uniform distribution over a small region
Training steps	Phase 1 steps	1.5×10^6
	Phase 2 steps	3×10^6
Other details	Maximum number of steps before an episode terminates	500
	Training steps between two evaluations	10^4
	Number of runs	5
	Number of evaluation episodes	10

Table 3: Hyperparameters of the deep Dyna-Q agent in MountainCarLoCA.

Neural networks	Transition model	MLP with *tanh*, $[64 \times 64 \times 63 \times 64 \times 64]$,
	Reward model	MLP with *tanh*, $[64 \times 64 \times 63 \times 64 \times 64]$,
	Termination model	MLP with *tanh*, $[64 \times 64 \times 63 \times 64 \times 64]$,
	Action-value estimator	MLP with *tanh*, $[64 \times 64 \times 64 \times 64]$,
Optimizer	Value optimizer	Adam, learning rate: 5×10^{-6}
	Model optimizer	Adam, learning rate: 5×10^{-5}
Other details	Exploration parameter	Epsilon greedy $\epsilon = 0.5$
	Number of random steps before training	50000
	Target network update frequency	500
	Number of model learning steps	5
	Number of planning steps	5
	Mini-batch size of model learning	32
	Mini-batch size of planning	32
	Replay buffer size	$4.5e6$

Table 4: Hyperparameters of the LOFO agent in MountainCarLoCA.

Embedding network architecture	MLP: $[64 \times 64 \times 64 \times 16]$, Activation Function: *tanh*
Optimizer	Adam, learning rate: 10^{-4}
β	10
Number of negative samples	128
Mini-batch size	32
Total number of random steps for creating dataset	100000
Number of training epochs	5
D_{local}	0.005
N_{local}	1

C.2 HYPERPARAMETERS

In our experiments, we have used similar deep Dyna-Q agents to the ones that were used in our MountainCarLoCA experiments: we only changed the neural network architecture for the model and the action-value estimator. Table 6

summarizes the architecture of the neural networks of these agents. Note that, differently from the MountainCarLoCA experiments, here we first encode a given state to a low-dimensional vector for the use of other parts of the model (transition, reward, and termination model). Then, we concatenate the given action with this vector and feed it to the MLPs. Finally, Table 7 summarizes the important hyperparameters of the REG agent. Note that these are the same hyperparameters that were used by Rahimi-Kalahroudi et al. (2023). For the PM-SimImp and PM-ScImp agents we have again just extended the REG agent in a way that is described in Sec. 5 and thus the hyperparameters are the same as the REG agent.

As the LOFO agent is also built on top of REG agent, the base hyperparameters of it is again the same as the REG agent and the additional hyperparameters of it can be reached at Table 8. Note again that these are the same hyperparameters that were used by Rahimi-Kalahroudi et al. (2023). For our neural embedding function, we have used the same hyperparameters with the LOFO agent.

Table 5: Experimental setup for the deep Dyna-Q agent in MiniGridLoCA.

Initial state distributions	Phase 1 training	Uniform distribution over the entire state space
	Phase 1 evaluation	Uniform distribution over the entire state space
	Phase 2 training	Uniform distribution over states within the red boundary (2×2 subgrid)
	Phase 2 evaluation	Uniform distribution over the entire state space
Training steps	Phase 1 steps Phase 2 steps	3×10^5 1.5×10^6
Other details	Maximum number of steps before an episode terminates Training steps between two evaluations Number of runs Number of evaluation episodes	100 10^4 5 10

Table 6: Neural network architecture for the deep Dyna-Q agent in MiniGridLoCA.

Neural networks	Transition model	CNN: (Channels: $[32 \times 64 \times 64]$ Kernel Sizes: $[8 \times 3 \times 3]$ Strides: $[4 \times 2 \times 2]$), Followed by Transposed CNN: (Channels: $[64 \times 32 \times 3]$ Kernel Sizes: $[6 \times 6 \times 5]$ Strides: $[1 \times 4 \times 3]$), Activation Function: *relu*
	Reward model	CNN: (Channels: $[32 \times 64 \times 64]$ Kernel Sizes: $[8 \times 3 \times 3]$ Strides: $[4 \times 2 \times 2]$), Followed by MLP: $[512]$, Activation Function: *relu*
	Termination model	CNN: (Channels: $[32 \times 64 \times 64]$ Kernel Sizes: $[8 \times 3 \times 3]$ Strides: $[4 \times 2 \times 2]$), Followed by MLP: $[512]$, Activation Function: *relu*
	Action-value estimator	CNN: (Channels: $[32 \times 64 \times 64]$ Kernel Sizes: $[8 \times 3 \times 3]$ Strides: $[4 \times 2 \times 2]$), Followed by MLP: $[512]$, Activation Function: *relu*

Table 7: Hyperparameters of the deep Dyna-Q agent in MiniGridLoCA.

Optimizer	Value optimizer	Adam, learning rate: 6.25×10^{-5}
	Model optimizer	Adam, learning rate: 10^{-4}
Other details	Exploration parameter	Epsilon greedy $\epsilon = 0.5$
	Number of random steps before training	2000
	Target network update frequency	5000
	Number of model learning steps	1
	Number of planning steps	1
	Mini-batch size of model learning	128
	Mini-batch size of planning	128
	Replay buffer size	$1.8e6$

Table 8: Hyperparameters of the LOFO agent in MiniGridLoCA.

Embedding network architecture	CNN: (Channels:$[32 \times 64 \times 64]$ Kernel Sizes:$[8 \times 3 \times 3]$ Strides:$[4 \times 2 \times 2]$), Followed by MLP:$[512 \times 16]$, Activation Function: *relu*
Optimizer	Adam, learning rate: 10^{-4}
β	10
Number of negative samples	128
Mini-batch size	32
Total number of random steps for creating dataset	25000
Number of training epochs	5
D_{local}	0.001
N_{local}	1

D Details of the ReacherLoCA Experiments

D.1 Experimental Setup and Hyperparameters

Tables 9 and 10 show the experimental setups that we used to evaluate the PlaNet and the Dreamer agents' adaptivity in ReacherLoCA, respectively.

For the hyperparameters of the PlaNet - REG and Dreamer - REG agents, we have used the same hyperparameter as in Rahimi-Kalahroudi et al. (2023) which consist of a replay buffer size that is equal to the total amount of training steps. For the PlaNet - PM-SimImp, Dreamer - PM-SimImp, PlaNet - PM-ScImp and Dreamer - PM-ScImp agents we have just extended the corresponding REG agents in a way that is described in Sec. 5 and thus the hyperparameters are the same as the corresponding REG agents.

As the PlaNet - LOFO and Dreamer - LOFO agents are also built on top of their corresponding REG agents, the base hyperparameters of them is again the same as the REG agents and the additional hyperparameters of them can be reached at Table 11. Note again that these are the same hyperparameters that were used by Rahimi-Kalahroudi et al. (2023). For our neural embedding function, we have used the same hyperparameters with the LOFO agent.

E Details of the RandomReacherLoCA Experiments

E.1 Experimental Setup and Hyperparameters

In the RandomReacherLoCA setup, the location of T1 (red) is randomly sampled from a circle that is centered in the middle of the environment (the dotted black circle in Fig. 7b of the main paper). And then, T2 (green) is placed at

Table 9: Experimental setup for the PlaNet agent in ReacherLoCA.

Initial state distributions	Phase 1 training	Uniform distribution over the entire state space
	Phase 1 evaluation	Uniform distribution over the entire state space
	Phase 2 training	Uniform distribution over states within the red boundary
	Phase 2 evaluation	Uniform distribution over the entire state space
Training steps	Phase 1 steps	3×10^5
	Phase 2 steps	2×10^5
Other details	Number of steps before an episode terminates	1000
	Training steps between two evaluations	15000
	Number of runs	5
	Number of evaluation episodes	5

Table 10: Experimental setup for the Dreamer agent in ReacherLoCA.

Initial state distributions	Phase 1 training	Uniform distribution over the entire state space
	Phase 1 evaluation	Uniform distribution over the entire state space
	Phase 2 training	Uniform distribution over states within the red boundary
	Phase 2 evaluation	Uniform distribution over the entire state space
Training steps	Phase 1 steps	10^6
	Phase 2 steps	10^6
Other details	Number of steps before an episode terminates	1000
	Training steps between two evaluations	10000
	Number of runs	5
	Number of evaluation episodes	8

Table 11: Hyperparameters of the LOFO agent in ReacherLoCA.

Embedding network architecture	CNN: (Channels: $[32 \times 64 \times 128 \times 256]$ Kernel Sizes: $[4 \times 4 \times 4 \times 4]$ Strides: $[2 \times 2 \times 2 \times 2]$), Followed by MLP: $[512 \times 64, 32]$, Activation Function: *relu*
Optimizer	Adam, learning rate: 10^{-4}
β	50
Number of negative samples	128
Mini-batch size	32
Total number of random steps for creating dataset	25000
Number of training epochs	5
D_{local}	0.05
N_{local}	10

the opposite end of this circle. Table 12 shows the experimental setup that we used to evaluate the Dreamer agents' adaptivity in RandomReacherLoCA.

For the hyperparameters of the PlaNet - REG and Dreamer - REG agents, we have again used the same hyperparameter as in Rahimi-Kalahroudi et al. (2023) which consist of a replay buffer size that is equal to the total amount of training steps. For the PlaNet - PM-SimImp, Dreamer - PM-SimImp, PlaNet - PM-ScImp and Dreamer - PM-ScImp agents we have again just extended the corresponding REG agents in a way that is described in Sec. 5 and thus the hyperparameters are the same as the corresponding REG agents.

As the PlaNet - LOFO and Dreamer - LOFO agents are also built on top of their corresponding REG agents, the base hyperparameters of them are again the same as the REG agents and the additional hyperparameters of them can be reached at Table 11. Note again that these are the same hyperparameters that were used by Rahimi-Kalahroudi et al. (2023). For our neural embedding function, we have used the same hyperparameters with the LOFO agent.

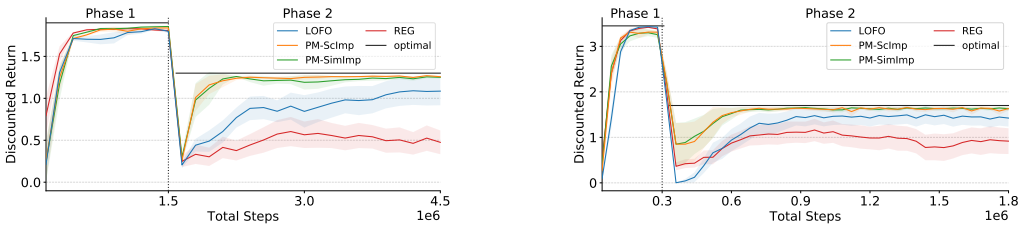
Table 12: Experimental setup the Dreamer agent in RandomReacherLoCA.

Initial state distributions	Phase 1 training	Uniform distribution over the entire state space
	Phase 1 evaluation	Uniform distribution over the entire state space
	Phase 2 training	Uniform distribution over states within the red boundary
	Phase 2 evaluation	Uniform distribution over the entire state space
Training steps	Phase 1 steps Phase 2 steps	1.5×10^6 3.5×10^6
Other details	Number of steps before an episode terminates Training steps between two evaluations Number of runs Number of evaluation episodes	1000 10000 5 8

F Experiments on More Challenging LoCA Setups

F.1 Deep Dyna-Q Experiments

The evaluation results of the deep Dyna-Q agents on the LoCA1 and LoCA2 setups of the MountainCar and MiniGrid domains are presented in Fig. 10 & 11, respectively.

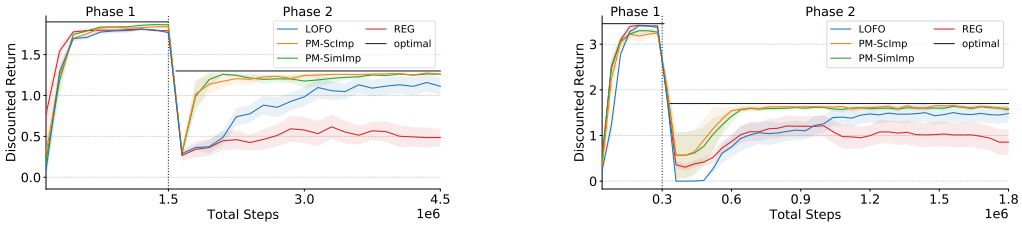


(a) MountainCarLoCA (b) MiniGridLoCA

Figure 10: Plots showing the learning curves of the deep Dyna-Q agents that are referred to as PM-SimImp, PM-ScImp, REG and LOFO on the (a) MountainCarLoCA1 s and (b) MiniGridLoCA1 setups. Each learning curve is an average discounted return over 20 runs and the shaded area represents the confidence intervals. The maximum possible return in each phase is represented by a solid black line.

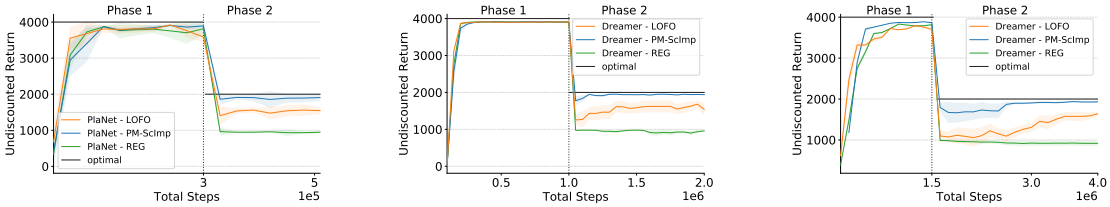
F.2 PlaNet and Dreamer Experiments

The evaluation results of the PlaNet and Dreamer agents on the LoCA1 and LoCA2 setups of the Reacher and RandomReacher domains are presented in Fig. 12 & 13, respectively.



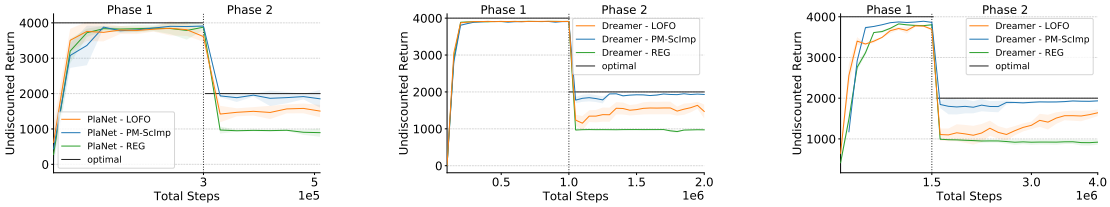
(a) MountainCarLoCA (b) MiniGridLoCA

Figure 11: Plots showing the learning curves of the deep Dyna-Q agents that are referred to as PM-SimImp, PM-ScImp, REG and LOFO on the (a) MountainCarLoCA2 s and (b) MiniGridLoCA2 setups. Each learning curve is an average discounted return over 20 runs and the shaded area represents the confidence intervals. The maximum possible return in each phase is represented by a solid black line.



(a) ReacherLoCA (b) ReacherLoCA (c) RandomReacherLoCA

Figure 12: Plots showing the learning curves of the PlaNet - PM-ScImp, Dreamer - PM-ScImp, PlaNet - REG, Dreamer - REG, PlaNet - LOFO, Dreamer - LOFO agents on the (a, b) ReacherLoCA1 and (c) RandomReacherLoCA1 setups. Each learning curve is an average undiscounted return over 10 runs and the shaded area represents the confidence intervals. The maximum possible return in each phase is represented by a solid black line.



(a) ReacherLoCA (b) ReacherLoCA (c) RandomReacherLoCA

Figure 13: Plots showing the learning curves of the PlaNet - PM-ScImp, Dreamer - PM-ScImp, PlaNet - REG, Dreamer - REG, PlaNet - LOFO, Dreamer - LOFO agents on the (a, b) ReacherLoCA2 and (c) RandomReacherLoCA2 setups. Each learning curve is an average undiscounted return over 10 runs and the shaded area represents the confidence intervals. The maximum possible return in each phase is represented by a solid black line.