Task-Specific Exploration in Meta-Reinforcement Learning via Task Reconstruction

Anonymous authors
Paper under double-blind review

Abstract

Reinforcement learning trains policies specialized for a single task. Meta-reinforcement learning (meta-RL) improves upon this by leveraging prior experience to train policies for few-shot adaptation to new tasks. However, existing meta-RL approaches often struggle to explore and learn tasks effectively. We introduce a novel meta-RL algorithm for learning to learn task-specific, sample-efficient exploration policies. We achieve this through task reconstruction, an original method for learning to identify and collect small but informative datasets from tasks. To leverage these datasets, we propose a meta-learned hyper-reward that encourages policies to learn to adapt. Empirical evaluations demonstrate that our algorithm adapts to a larger variety of tasks and achieves higher returns than existing meta-RL methods. Additionally, we show that even with full task information, adaptation is more challenging than previously assumed. However, policies trained with our hyper-reward adapt to new tasks successfully.

1 Introduction

Research in reinforcement learning (RL) has led to many impressive achievements and powerful methods in the past decade (Sutton, 2018). However, real-world usage is still not widespread, as the algorithms used to solve RL problems often suffer from low sample efficiency and poor generalization to new tasks (Hospedales et al., 2021; Beck et al., 2023b). Meta-reinforcement learning (meta-RL) is a re-emerging approach that tackles these issues. It follows the meta-learning approach of 'learning to learn' (Schmidhuber, 1987; Thrun & Pratt, 1998), making it well-suited for few-shot adaptation. In few-shot adaptation settings, the goal is to become optimal in any new task from a given distribution of tasks, after collecting only a few episodes. Instead of directly learning to solve tasks from scratch, a meta-RL agent finds RL algorithms that quickly learn the optimal policy. While meta-RL agents are general, the algorithms they produce are task-specific. These meta-learned algorithms contain prior knowledge of the task distribution. By leveraging this prior, they quickly learn the optimal method of exploring or solving a task. Therefore, for any task in the task distribution, meta-learned algorithms are expected to be more sample-efficient than and outperform standard RL algorithms.

Meta-RL has already been successfully applied to ad hoc teamwork (Zintgraf et al., 2021a; He et al., 2023; Mirsky et al., 2022), robotics (Nagabandi et al., 2018; Zhao et al., 2022), human-robot interaction (Gao et al., 2019; Ballou et al., 2023), multi-agent RL (Yang et al., 2022; Xu et al., 2022), and sim-to-real transfer (Arndt et al., 2020). Despite this, meta-RL has not yet fully addressed the challenges it aims to overcome (Stoican et al., 2023). While recent years have seen several improvements, meta-RL algorithms still struggle with adapting to complex tasks. A significant issue is the difficulty of designing sample-efficient exploration strategies for solving complex, dynamic RL environments. The advantage of meta-RL is that these strategies do not have to be manually crafted, but can be learned from data for each particular task distribution. Often, meta-RL agents interact with a task for a few episodes, before being evaluated on that same task. These agents can be seen as exploring the task, using an implicit or explicit meta-learned exploration strategy. However, implicit exploration can be too sample inefficient for few-shot adaptation (e.g. the agent is not effectively using its prior of the task distribution during exploration). Recent approaches have shown that explicitly optimizing for sample-efficient exploration leads to more powerful few-shot adaptation agents (Beck et al., 2023b).

We introduce ***La**tent **S**pace **E**xploration via Task **R**econstruction (**LaSER**)*, a novel meta-RL algorithm for few-shot adaptation.[1] LaSER's primary objective is to learn task-specific exploration strategies. The core idea of our method is to identify small datasets that capture rich task-specific information with minimal redundancy. To achieve this, we introduce a novel approach of meta-learning by linearly reconstructing these small datasets into much larger datasets collected from the same task. We refer to this process as task reconstruction. It serves as a measure of how effective a given small dataset is for few-shot adaptation. Following this measure, LaSER meta-learns a latent space for directly identifying such small datasets, without having to first collect a larger one. This allows training a sample-efficient exploration policy that can collect task-specific data online. Ultimately, this data is encoded into compact task descriptors, used to train a policy that solves tasks. However, we show that previous approaches to training this policy fail even if there exists an oracle that provides optimal task representations. As a result, LaSER's secondary objective is to learn to use task representations efficiently. To achieve this, we propose a new method of augmenting the standard RL objective. We summarize our main contributions as follows.

1. We formalize two important assumptions about the data required for few-shot adaptation. These assumptions form the basis of our task reconstruction method. We then show how an encoder, built from these ideas, can learn representations that facilitate exploration.
2. We leverage these representations to design a simple intrinsic reward for training a task-specific, sample-efficient exploration policy.
3. We augment the standard RL objective with a term that encourages the effective use of task descriptors for solving new few-shot tasks optimally. This simple change allows meta-RL policies to be trained with standard RL algorithms.
4. We introduce a hybrid encoder architecture composed of a unidirectional and a bidirectional model. The former is used for online task exploration. The latter offers richer representations of exploration data.

We formalize the meta-RL problem and introduce the necessary background in Sec. 2. In Sec. 3 we present our approach to the objectives described in this section, and introduce LaSER, our novel meta-RL algorithm. We position our work within the meta-RL literature in Sec. 4. We provide empirical results in Sec. 5, where we evaluate LaSER against existing meta-RL algorithms. We first assess LaSER's performance and adaptability to new tasks after a short exploration phase (i.e. four episodes). We then analyze each individual component, i.e. exploration policy, encoder, and task-solving policy. Finally, we conclude and provide directions for future research in Sec. 6.

2 Preliminaries

Meta-Reinforcement Learning. Meta-RL extends the standard RL problem. Instead of learning from a single task, a meta-RL agent trains on a distribution of tasks. Usually, the agent's objective is adaptation to new tasks. We consider a probability distribution over tasks and define each task $\mathcal{M}_i \sim p(\mathcal{M})$ as a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, T_i, R_i, \gamma, H)$. All MDPs share the set of states $\mathcal{S}$, set of actions $\mathcal{A}$, discount factor γ, and horizon H. However, the transition function, given as the probability $T_i(s' \mid s, a, \mathcal{M}_i)$ of transitioning from state s to state s' by taking action a in task $\mathcal{M}_i$, is task-dependent. Similarly, the reward function R_i is task-dependent. An episode $\tau = \{s_t, a_t, r_t\}_{t=1}^{H}$ is a sequence of states, actions, and rewards, such that at each time-step t a tuple (s_t, a_t, r_t) is collected. For a distribution over episodes $P^{\pi}_{\mathcal{M}_i}(\tau)$, we denote $\tau \sim P^{\pi}_{\mathcal{M}_i}$ to be an episode collected by following a policy $\pi(a \mid s)$ in task $\mathcal{M}_i$. The return of an episode τ is the accumulated discounted reward $G(\tau) = \sum_{t=1}^{H} \gamma^t r_t$.

---

[1] Our code is publicly available in an anonymized repository at https://anonymous.4open.science/r/LaSER-anon-4BBB.

Meta-RL for Few-Shot Adaptation. In the context of few-shot adaptation, meta-RL can be used to optimize a task policy π to solve any previously unseen task $\mathcal{M}_i \sim p(\mathcal{M})$. The few-shot constraint is that π must adapt using only K episodes collected from $P^{\pi^{\text{explore}}}_{\mathcal{M}_i}$, for some exploratory policy π^{explore}. We call this a K-shot adaptation problem. As in Beck et al. (2023b), we define a sequence of K episodes to be a meta-episode $\mathcal{D}^{(K)}_i = \{\tau_k\}^K_{k=1}$. A common way to adapt is by using an encoder g to compute a latent task descriptor $z_i = g(\mathcal{D}^{(K)}_i)$. Task-specific actions can be generated by conditioning π on z_i. Importantly, z_i should contain information about the task dynamics, i.e. T_i and R_i, to allow generalization to new tasks. To define the meta-RL objective, we follow Liu et al. (2021) and decouple the task policy π from the exploration policy π^{explore}. Contrary to π, the goal of π^{explore} is to collect an informative meta-episode, without necessarily achieving a high return. The exploration policy π^{explore} can be made task-specific by conditioning it on a task representation Γ_i. Here, Γ_i is either the same as z_i or a new representation. Finally, according to Liu et al. (2021), meta-RL aims to maximize the objective

$$J(\pi, \pi^{\text{explore}}) = \mathbb{E}_{\mathcal{M}_i \sim p(\mathcal{M}), \mathcal{D}^{(K)}_i \sim P^{\pi^{\text{explore}}}_{\mathcal{M}_i}} \left[V^{\pi}_i(z_i) \right], \tag{1}$$

where $V^{\pi}_i(z_i) = \mathbb{E}_{\tau \sim P^{\pi(\cdot|z_i)}_{\mathcal{M}_i}} [G(\tau)]$ is the expected return in task $\mathcal{M}_i$, for π conditioned on z_i.

Training Setup. Meta-training is the process of training a meta-RL agent. The goal is to learn a sample-efficient algorithm for optimizing policies for each task encountered from $p(\mathcal{M})$. The agent's ability to generalize is evaluated during meta-testing, by solving new tasks from $p(\mathcal{M})$. For each meta-testing task, the agent is first allowed to collect and learn from K episodes. Then, it is evaluated according to Eq. 1. We refer to collecting K episodes from $P^{\pi^{\text{explore}}}_{\mathcal{M}_i}$ as task exploration. This type of exploration is meta-learned to be task-specific. Besides task exploration, the agent still performs the standard RL exploration. To achieve their objectives, π and π^{explore} have to explore their environment during meta-training. This is commonly referred to as meta-exploration (Beck et al., 2023b). During meta-testing, there is no meta-exploration.

Data Representation. States, actions, and rewards can be represented as vectors or scalars. So, a timestep (s_t, a_t, r_t) is a d-dimensional vector, where d is the sum of the dimensions of s_t, a_t, and r_t. We extend this to episodes. For $H' = Hd$, an episode $\tau \in \mathbb{R}^{H'}$ is a vector of H concatenated timesteps. We can further extend this to meta-episodes. For a meta-episode $\mathcal{D}^{(K)} \in \mathbb{R}^{H' \times K}$ composed of K episodes, the k-th column $\mathcal{D}^{(K)}_{:,k}$ corresponds to the k-th episode. From this point, episodes and meta-episodes are always represented as tensors instead of sequences.

3 Latent Space Exploration via Decomposed Representations

In this section, we present the main components of LaSER. We train two distinct policies: π for task-solving and π^{explore} for task exploration. Additionally, we introduce an encoder, g, designed to encode data collected by π^{explore}. We use a transformer architecture of size d_{model} to model g, taking advantage of transformers' strengths in in-context learning and long-sequence modeling.

Given a descriptor z_i of a task $\mathcal{M}_i$, Sec. 3.1 introduces a simple novel term to be added to the standard RL objective. This term encourages exploration whenever π can improve its performance by understanding z_i better. Our approach to task exploration, i.e. the process of collecting the data required to compute z_i, is presented in Sec. 3.2. This section introduces a second encoder, f_u, modeled as a unidirectional transformer of size d_{model}. The goal of f_u is to train π^{explore} for task-exploration. Sec. 3.3 introduces a novel method of optimizing g and f_u for sample-efficient task-specific exploration. We introduce important assumptions on the data collected for few-shot adaptation, then show how a practical algorithm can be built from them. Finally, we describe our proposed architecture in Sec. 3.4, and our meta-training and meta-testing algorithms in Sec. 3.5. To simplify notation, we drop the task subscript i whenever it is clear we are referencing task-specific data or representations.

3.1 Solving Tasks

We have already established that task exploration is one of the most vital issues in meta-RL, and simultaneously the main topic of our work. However, consider a meta-RL agent that explores tasks optimally.

Such an agent is not guaranteed to be optimal in a new task $\mathcal{M}_i \sim p(\mathcal{M})$. This is because optimizing the task-solving policy π is non-trivial. Inspired by the field of representation learning in RL (Igl et al., 2018; Papoudakis et al., 2021), many recent works in meta-RL simply augment the current state s_t with a task descriptor or belief z, and then optimize $\pi(a_t \mid s_t, z)$ using standard RL algorithms. This approach might not generalize well in complex task distributions, even if z fully describes the task $\mathcal{M}_i$ (Beukman et al., 2024). Beck et al. (2023a) and Beukman et al. (2024) propose to solve this issue by using a hypernetwork (Ha et al., 2017) conditioned on z to generate the weights of a context-dependent policy π. In our experiments, we have found this approach to only offer a marginal improvement in the agent's generalization, while coming with a considerable computational cost. We introduce an alternative that does not require hypernetworks. Our approach augments the standard RL reward with a term that encapsulates performance in the meta-RL setting. By using this new reward, we can simply optimize π using standard RL algorithms.

We start from the hypothesis that π is sub-optimal because it fails to understand that z offers important information about the current task $\mathcal{M}_i$. In a standard RL setting, a policy $\pi(a_t \mid s_t)$ would need to explore enough to understand the (single) task it is expected to solve. In our meta-RL setting, we have an additional exploration objective. The policy $\pi(a_t \mid s_t, z)$ must understand the relationship between s_t and z, at any timestep t where the optimal action depends on both s_t and z. In other words, π needs to explore more than in a standard RL setting.

Following this idea, we propose a method that encourages π to explore more if the task descriptor z is not properly used. Consider the value function $V(s_t, z)$. Equivalently, consider $V(s_t, \cdot)$, which computes the value at state s_t without considering z. We make the assumption that, for an optimal policy, $V(s_t, \cdot) \leq V(s_t, z)$ should hold at any timestep t. That is, using z should never decrease performance. If this does not hold for some state s_t, then the agent should explore from s_t. We implement this idea as a hyper-reward

$$r_t^+ = r_t + \beta\, w(s_t, z)\, S[\pi](s_t, z), \tag{2}$$

$$\text{where } w(s_t, z) = \max\left(0, \tanh\left(V(s_t, \cdot) - V(s_t, z) - \zeta\right)\right). \tag{3}$$

Here, r_t is the environment reward, β is a constant, S is the entropy of $\pi(a_t \mid s_t, z)$, and $w(s_t, z)$ is a dynamic weight on this entropy. Since increasing entropy is equivalent to increasing exploration, we design $w(s_t, z)$ to be non-zero only when $V(s_t, \cdot) - V(s_t, z) \geq \zeta$, for a given threshold $\zeta \in (-\infty, 0]$.

Intuitively, $w(s_t, z) > 0$ means that the agent is disregarding the information provided by z, and should explore more from state s_t. When $w(s_t, z) = 0$, i.e. $V(s_t, \cdot) \leq V(s_t, z) + \zeta$, we recover the original RL objective, with $r_t^+ = r_t$. By simply replacing r_t with r_t^+, π can now be optimized using standard RL algorithms. We use PPO (Schulman et al., 2017) in our work. Additionally, we stabilize PPO (Sun et al., 2022; 2023; Moalla et al., 2024) using proximal feature optimization (PFO) (Moalla et al., 2024). In Appendix A, we provide a short overview of PPO, and then explain our method for applying it to meta-RL.

This section introduced a novel meta-exploration method for meta-training task policies. However, it does not address meta-learning task exploration. We tackle this distinct challenge in the next subsection.

3.2 Exploring Tasks

The objective of the exploration policy π^{explore} is to collect a single, informative, meta-episode $\mathcal{D}^{(K)}$ for a task $\mathcal{M}_i \sim p(\mathcal{M})$. Specifically, π^{explore} should collect information about the task-specific dynamics of $\mathcal{M}_i$, while avoiding irrelevant or redundant exploration. Ultimately, $\mathcal{D}^{(K)}$ should enable a task policy π to become optimal in $\mathcal{M}_i$, assuming π has enough prior knowledge about the task distribution $p(\mathcal{M})$.

The encoder f_u computes $\Gamma = f_u(\mathcal{D}^{(K)}) \in \mathbb{R}^{(Hd_{\text{model}}) \times K}$. Here, Γ is a latent representation of the K episodes in $\mathcal{D}^{(K)}$, with $\Gamma_{:,k}$ denoting the representation of the k-th episode $\mathcal{D}^{(K)}_{:,k}$. We define f_u such that the similarity between $\Gamma_{:,k}$ and $\Gamma_{:,k'}$ is inversely proportional to how useful it is to collect episode $\mathcal{D}^{(K)}_{:,k'}$, after having already collected episode $\mathcal{D}^{(K)}_{:,k}$, for any $k, k' \in [K], k < k'$.[2] We measure similarity as the inner product $\Gamma^T\Gamma \in \mathbb{R}^{K \times K}$. Lower scores correspond to lower similarity between the K episodes, which in turn

---

[2] With a slight abuse of notation, we use $[N]$ to mean the sequence $[1, 2, \ldots, N]$ for any integer N.

corresponds to $\mathcal{D}^{(K)}$ being more informative. In Sec. 3.3, we show how to train a function f_u with this property.

We use this idea to optimize π^{explore} by computing the average similarity in $\mathcal{D}^{(K)}$ as a vector $\boldsymbol{d} = \frac{1}{K}(\Gamma^T\Gamma)^T\mathbf{1} \in \mathbb{R}^K$. Here, $\boldsymbol{d}_k$ represents the average similarity between the k-th episode and all other episodes. A meta-episode contains an informative and diverse set of episodes if $\boldsymbol{d} \approx 0$. Based on this, we define an intrinsic reward function for encouraging task exploration. For the k-th exploration episode, the agent receives a task-dependent reward

$$\tilde{R}_k(s_t^k, a_t^k) = \begin{cases} \exp\left(-\frac{1}{\sigma}\boldsymbol{d}_k^2\right) & \text{if the } k\text{-th episode terminates at timestep } t, \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where σ is a constant, and s_t^k and a_t^k denote the state and action, respectively, at timestep t of episode k. We use PPO to train π^{explore} to maximize this intrinsic reward. In practice, π^{explore} is trained on full meta-episodes. Therefore, we use a causal mask when computing $\boldsymbol{d}$, to hide the similarity between any episodes k and k' when $k < k'$.

For π^{explore} to collect a useful episode, it must have knowledge of the episodes it has already collected. We denote $\mathcal{D}^{(:k,:t)}$ to be an incomplete meta-episode, containing $k-1$ complete episodes, and the first t timesteps of the k-th episode, for $k \in [K], t \in [H]$. We then condition π^{explore} on the history $\Gamma^{(:k,:t)} = f_u(\mathcal{D}^{(:k,:t)})$ and explore episode k by taking actions $a_{t+1}^k \sim \pi^{\text{explore}}(\cdot \mid s_{t+1}^k, \Gamma^{(:k,:t)})$ at each timestep. The overall task exploration process is shown visually in Fig. 1.

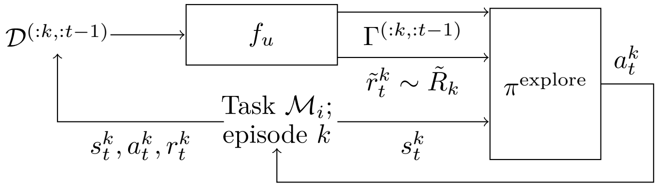

$\mathcal{D}^{(:k,:t-1)}$
f_u
$\Gamma^{(:k,:t-1)}$
$\tilde{r}_t^k \sim \tilde{R}_k$
π^{explore}
a_t^k
Task $\mathcal{M}_i$; episode k
s_t^k, a_t^k, r_t^k
s_t^k

Figure 1: Task exploration. The policy π^{explore} explores a task $\mathcal{M}_i$ by collecting K episodes. At each timestep t of each episode k, π^{explore} is conditioned on the current state s_t^k and the latent representation $\Gamma^{(:k,:t-1)}$ of previous timesteps and episodes. Its objective is to take actions that maximize the intrinsic exploration reward function $\hat{R}_k$. Both $\Gamma^{(:k,:t-1)}$ and $\hat{R}_k$ are computed by the encoder f_u from the data collected.

3.3 Learning by Reconstructing Tasks

The final components of our meta-RL algorithms are encoders g and f_u, which enable task-solving and task exploration, respectively. Let $\mathbf{B} \in \mathbb{R}^{Q\times H'\times K}$ be the tensor of Q meta-episodes collected from $\mathcal{M}_i$. That is, for $j \in [Q]$, $\mathbf{B}_{j,:,:}$ is the j-th meta-episode in $\mathbf{B}$. For simplicity, we use $\mathbf{B}_j$ to denote $\mathbf{B}_{j,:,:}$. Additionally, to emphasize the relationship between $\mathbf{B}_j$ and $\mathbf{B}$, we refer to meta-episodes as $\mathbf{B}_j$ instead of $\mathcal{D}^{(K)}$.

Recall that, given $\mathbf{B}_j$, we use $z = g(\mathbf{B}_j)$ for task-solving and $\Gamma = f_u(\mathbf{B}_j)$ for task exploration. The LaSER encoders g and f_u are optimized based on a key assumption about the data used for few-shot adaptation. First, we formalize a (weak) assumption necessary for K-shot adaptation to be feasible.

Assumption 1 (K-Shot Adaptation)**.** *A policy π is K-shot adaptable in a distribution $p(\mathcal{M})$ over MDPs if, for any $\mathcal{M}_i \sim p(\mathcal{M})$, π becomes optimal in $\mathcal{M}_i$ after at most K episodes collected from $\mathcal{M}_i$.*

Finding these K episodes can be difficult. Instead, consider rearranging the tensor $\mathbf{B} \in \mathbb{R}^{Q\times H'\times K}$ into a matrix $\boldsymbol{B}_{[2]} \in \mathbb{R}^{H'\times(QK)}$ via mode-2 matricization (Vasilescu, 2009). That is, $(\boldsymbol{B}_{[2]})_{:,k}$ represents the k-th episode, for $k \in [QK]$. If a policy can become optimal in a K-shot task $\mathcal{M}_i$ using the QK episodes in $\boldsymbol{B}_{[2]}$, Assumption 1 implies that $\boldsymbol{B}_{[2]}$ contains the K episodes required for K-shot adaptation. More precisely, there exists a meta-episode $\boldsymbol{B}^*_{[2]}$ composed of these K episodes. To create a practical algorithm from this idea, we make the following assumption on the relationship between $\boldsymbol{B}_{[2]}$ and $\boldsymbol{B}^*_{[2]}$.

Assumption 2 (Linear Task Reconstruction)**.** *Let $\boldsymbol{B}_{[2]} \in \mathbb{R}^{H' \times (QK)}$ contain QK episodes collected from an MDP $\mathcal{M}_i$ such that $\boldsymbol{B}_{[2]}$ is sufficient for K-shot adaptation. Let $\boldsymbol{B}^*_{[2]} \in \mathbb{R}^{H' \times K}$ be the submatrix of $\boldsymbol{B}_{[2]}$ that contains these K necessary episodes. Then, we assume there exists $\boldsymbol{C}_{[2]} \in \mathbb{R}^{K \times (QK)}$ such that*

$$\boldsymbol{B}^*_{[2]} \boldsymbol{C}_{[2]} \approx \boldsymbol{B}_{[2]}. \tag{5}$$

It follows from Assumption 2 that $\text{rank}(\boldsymbol{B}_{[2]}) \approx K$. Let $\boldsymbol{B}^*_{[2]}$ be the submatrix containing K linearly independent columns of $\boldsymbol{B}_{[2]}$. Then, $\boldsymbol{B}^*_{[2]}$ is an optimal full-rank approximation of $\boldsymbol{B}_{[2]}$, and there exists a coefficients matrix $\boldsymbol{C}_{[2]}$ such that Eq. 5 holds. For efficiency and simplicity, we switch back to tensor representations. We use $\mathbf{B}$ instead of $\boldsymbol{B}_{[2]}$, represent $\boldsymbol{B}^*_{[2]}$ as the meta-episode $\mathbf{B}_j \in \mathbb{R}^{H' \times K}$ for a given $j \in [Q]$, and rearrange $\boldsymbol{C}_{[2]}$ into $\mathbf{C} \in \mathbb{R}^{Q \times K \times K}$. Eq. 5 is then reframed as the batch matrix multiplication $\mathbf{B}_{j,:,:} \mathbf{C}_{l,:,:} \approx \mathbf{B}_{l,:,:}$ for all $l \in [Q]$, which we denote compactly as $\mathbf{B}_j \mathbf{C} \approx \mathbf{B}$.

To identify meta-episodes of interest during task exploration, we introduce a novel approach of reconstructing tasks, based on Assumption 2. For a meta-episode $\mathbf{B}_j$ collected by π^{explore}, our method allows us to quantify its effectiveness for K-shot adaptation. That is, for a given j, we measure how effective it is to compute a task descriptor z from $\mathbf{B}_j$ instead of $\mathbf{B}$. Following Eq. 5, the problem is reduced to probing the linear relationship between $\mathbf{B}_j$ and $\mathbf{B}$, by attempting to find coefficients $\mathbf{C}$. To this end, we define the task-reconstruction loss

$$\mathcal{L}_{\text{t-rec}} = \mathbb{E}_{\mathcal{M}_i \sim p(\mathcal{M}),\, \mathbf{B} \sim P^{\pi^{\text{explore}}}_{\mathcal{M}_i},\, j \sim U(Q)} \left[\text{MSE} \left(\mathbf{B}_j \mathbf{C},\, \mathbf{B} \right) \right], \tag{6}$$

where $U(Q)$ is a uniform distribution over the integers $\{1, 2, \dots, Q\}$, MSE is the mean squared error, and $\mathbf{C}$ is computed from $\mathbf{B}$ and $\mathbf{B}_j$, using encoder g (see Sec. 3.4). By minimizing this loss, the agent learns to find $\mathbf{C}$. This is then used to assess the quality of the meta-episode $\mathbf{B}_j$.

Note that computing $\mathbf{C}$ is equivalent to finding an approximate solution to the system of linear equations $\mathbf{B}_j \mathbf{C} = \mathbf{B}$. Therefore, g requiring $\mathbf{B}$ as an input is an unavoidable constraint. Consequently, $\mathbf{C}$ cannot be computed during task exploration, where the agent is limited to collecting a single meta-episode. We present an alternative approach for optimizing f_u for sample-efficient exploration. Consider a target

$$\delta_j = 1 - \exp \left(-\xi \overline{\left(\mathbf{B}_j \hat{\mathbf{C}} - \mathbf{B} \right)^2} \right), \tag{7}$$

where ξ is a constant and $\overline{\mathbf{T}}$ gives the element-wise mean of any tensor $\mathbf{T}$. We compute $\hat{\mathbf{C}}$ similarly to $\mathbf{C}$, but use a target network to improve training stability. Having $\delta_j \approx 0$ means $\mathbf{B}_j$ is a good approximation of $\mathbf{B}$. We therefore optimize f_u to learn a latent space that encodes this information. Specifically, for a given Γ, we optimize the similarity between any two episode representations $\Gamma_{:,k}$ and $\Gamma_{:,k'}$ to be δ_j, for $k, k' \in [K]$. As in Sec. 3.2, we use the inner product to measure similarity. Inspired by contrastive learning, we use δ_j to define the loss

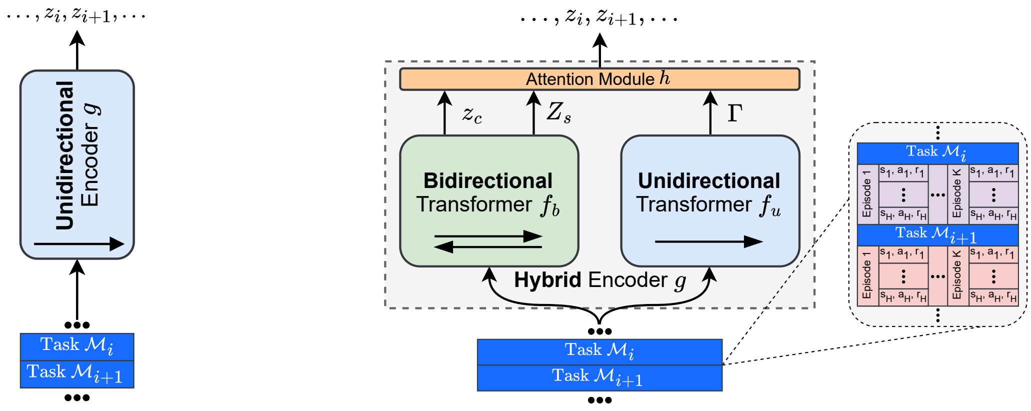
$$\mathcal{L}_{\text{contr}} = \mathbb{E}_{\mathcal{M}_i \sim p(\mathcal{M}),\, \mathbf{B} \sim P^{\pi^{\text{explore}}}_{\mathcal{M}_i},\, j \sim U(Q)} \left[\text{MSE} \left(\Gamma^T \Gamma,\, \boldsymbol{A}(\delta_j) \right) \right], \tag{8}$$
$$\text{where } \Gamma = f_u(\mathbf{B}_j).$$

Here, $\boldsymbol{A}(\delta_j) \in \mathbb{R}^{K \times K}$ is the matrix in which off-diagonal elements are δ_j and diagonal elements are 1. Intuitively, collecting a meta-episode where all vectors $\{\Gamma_{:,k}\}_{k=1}^K$ are orthogonal to each other is now equivalent to searching for a $\mathbf{B}_j$ that closely approximates $\mathbf{B}$. The important advantage is that our method can be used for K-shot adaptation. The alternative approach of finding $\mathbf{B}_j$ by collecting Q meta-episodes would be much more sample-inefficient.

3.4 Architecture and Optimization

Next, we introduce the architecture of our encoder g. Because of the close relation between g and f_u, we define f_u to be a sub-network of g (i.e. g and f_u are optimized together). Recall that f_u separates informative meta-episodes from non-informative ones during K-shot adaptation. Moreover, f_u is constrained to encode meta-episodes online, timestep by timestep, as they are being collected. To satisfy this, we model f_u as a unidirectional transformer with parameters ω_u and train it using autoregressive attention masks (Vaswani et al., 2017). Many previous works use a similar unidirectional encoder (Mishra et al., 2018; Rakelly et al., 2019; Zintgraf et al., 2021b; Melo, 2022). An obvious limitation is that only interactions between the current and past timesteps are considered. Future timesteps could also possess useful information, leading to richer representations (Devlin et al., 2019; Banino et al., 2015). Therefore, we define an additional bidirectional transformer f_b, with parameters ω_b. Finally, we set g to be the hybrid composed of f_u and f_b.

Given $\mathbf{B}_j$, f_b computes two new latent representations $\boldsymbol{z}_c, \boldsymbol{Z}_s = f_b(\mathbf{B}_j; \omega_b)$. Importantly, f_b is bidirectional, so it encodes data "offline", i.e. after task exploration is over. To capture the task-independent structure shared by all MDPs in $p(\mathcal{M})$, $\boldsymbol{z}_c \in \mathbb{R}^{d_{\text{model}}}$ is computed from multiple tasks. Conversely, $\boldsymbol{Z}_s \in \mathbb{R}^{HK \times d_{\text{model}}}$ is a task-specific representation of $\mathcal{M}_i$, encoded from a meta-episode $\mathbf{B}_j$ in $\mathbf{B}$. While $\boldsymbol{Z}_s$ and Γ have similar roles, Γ might fail to capture structure that is irrelevant for task exploration, but important for task-solving. We compute the final task descriptor z as a combination of $\boldsymbol{z}_c$, $\boldsymbol{Z}_s$, and Γ, using a function h with parameters ω_h. In practice, h consists of two attention layers. We compute $z \in \mathbb{R}^{HK \times d_{\text{model}}}$ as $z = g(\mathbf{B}_j; \omega) = h(\boldsymbol{z}_c, \boldsymbol{Z}_s, \Gamma; \omega_h)$, for g_ω with concatenated parameters $\omega = \omega_u \oplus \omega_b \oplus \omega_h$. Our architecture can be seen in Fig. 2.



(a) (b)

Figure 2: (a) Encoders used for meta-RL are usually unidirectional. They process task exploration data online, step by step, as it is being collected. (b) Our architecture enhances this approach by adding a bidirectional encoder. This can be used offline after task exploration is complete. This addition leads to more informative representations.

To optimize g_ω, we define two pre-training heads $h_{\text{t-rec}}$ and h_{rec}, parameterized by $\omega_{\text{t-rec}}$ and ω_{rec}, respectively. The former is used to compute the tensor of coefficients required in Eq. 6, i.e. $\mathbf{C} = h_{\text{t-rec}}(\boldsymbol{Z}_s, \Gamma; \omega_{\text{t-rec}})$. The target $\hat{\mathbf{C}}$ is computed similarly, but uses parameters $\hat{\omega}_{\text{t-rec}}$ instead. Similarly to Mnih et al. (2015), to increase stability, $\hat{\omega}_{\text{t-rec}}$ is updated to $\omega_{\text{t-rec}}$ only every ν iterations. The latter head is commonly used in bidirectional transformers to reconstruct the input data. Since these operations are only required during meta-training, the heads are replaced by policies π^{explore} and π during meta-testing. Fig. 3 gives an overview of the heads.

Bidirectional transformers are commonly trained through masked self-supervision (Devlin et al., 2019; Lewis et al., 2019), using a reconstruction loss $\mathcal{L}_{\text{rec}}$. After adding noise to the input, the encoder learns useful representations by reconstructing the denoised input (see Appendix B for more details). We extend this and define the loss of the encoder g_ω as

$$\mathcal{L}_{\text{LaSER}}(\omega, \omega_{\text{rec}}, \omega_{\text{t-rec}}) = c_{\text{rec}}\mathcal{L}_{\text{rec}}(\omega, \omega_{\text{rec}}) + c_{\text{t-rec}}\mathcal{L}_{\text{t-rec}}(\omega, \omega_{\text{t-rec}}) + c_{\text{contr}}\mathcal{L}_{\text{contr}}(\omega) + c_{\mathcal{R}}\mathcal{R}(\omega)\,, \tag{9}$$

where $c_{\text{rec}}, c_{\text{t-rec}}, c_{\text{contr}}, c_{\mathcal{R}}$ are coefficients, and $\mathcal{R}$ is a regularization term. It regularizes the latent spaces generated by f_b and f_u. Following Piratla et al. (2020), $\mathcal{R}$ enforces a soft orthogonality constraint between the vector representations $\boldsymbol{z}_c$ and $(\boldsymbol{Z}_s)_{t,:}$, at each timestep $t \in [HK]$, by minimizing $(\boldsymbol{z}_c \cdot (\boldsymbol{Z}_s)_{t,:})^2$. Intuitively, $\boldsymbol{Z}_s$

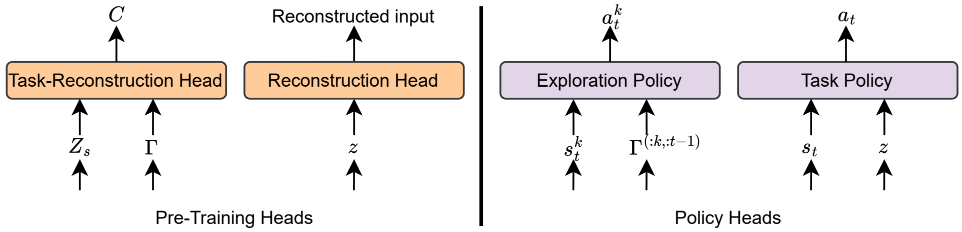


Figure 3: (Left) Heads used to pre-train the encoder. (Right) Policy heads are used for task exploration and task exploitation, respectively.

is expected to focus on task-specific structure, not capture information already contained in $\boldsymbol{z}_c$. Additionally, $\mathcal{R}$ normalizes the vector representations $\boldsymbol{z}_c$, $(\boldsymbol{Z}_s)_{t,:}$, and $\Gamma_{:,k}$, for all $t \in [HK], k \in [K]$.

3.5 Meta-Training and Meta-Testing

The LaSER meta-training algorithm is shown in Alg. 1. It is composed of two training phases. We first train the exploration policy $\pi_\phi^{\text{explore}}$ parameterized by ϕ, and the encoder g_ω parameterized by ω. Note that these components are trained together because they are interdependent. We alternate between optimizing one while keeping the other fixed. In the second stage, these two components are fixed and only the task policy π_θ, parameterized by θ, is optimized.

Recall that $\boldsymbol{z}_c$ is computed from multiple tasks. This is only feasible during pre-training. Therefore, we compute a fixed $\hat{\boldsymbol{z}}_c$ from the pre-training data. We use $\hat{\boldsymbol{z}}_c$ to train the task policy and perform meta-testing. During meta-testing, for $\mathcal{M}_i \sim p(\mathcal{M})$, the meta-trained agent first collects K episodes and computes z. Then, it uses z to find the optimal policy for $\mathcal{M}_i$. This is shown in Alg. 2.

4 Related Work

Meta-RL Some of the earlier successes in meta-RL involved algorithms like MAML (Finn et al., 2017) and RL^2 (Duan et al., 2016; Wang et al., 2016). Gradient-based methods like MAML and related algorithms (Sung et al., 2017; Li et al., 2017; Zintgraf et al., 2019; Gupta et al., 2018) adapt to tasks through an inner loop and meta-learn by optimizing across tasks in an outer loop. RL^2, on the other hand, makes use of recurrent neural networks to obtain useful representations of tasks. The approach of first identifying a task, then adapting a task-specific optimal policy, is commonly known as context-based meta-RL. Recent techniques, including our method, follow this paradigm. Context-based approaches are often more sample-efficient than other types of meta-RL algorithms, making them ideal for few-shot adaptation (Beck et al., 2023b). Key methods for identifying tasks include meta-learning latent representations of value functions (Rakelly et al., 2019) or MDP dynamics (Zhou et al., 2019; Zintgraf et al., 2021b;c), model-based meta-RL (Clavera et al., 2018; Nagabandi et al., 2018), hybrid techniques that combine context-based with gradient-based methods (Imagawa et al., 2022), using permutation variant and invariant sequence models (Beck et al., 2024), or enhancing tasks by incorporating language instructions (Bing et al., 2023). Attention mechanisms (Bahdanau et al., 2015) and transformers (Vaswani et al., 2017) have also been adopted by the meta-RL community. Their success in in-context learning, long-sequence modeling, and efficient parallelizable training, makes them align well with the needs of context-based meta-RL. Earlier research focused on in-context meta-learning through attention (Mishra et al., 2018), while more recent work used transformer architectures (Melo, 2022; Shala et al., 2024). Finally, after identifying a task, the task-solving policy is usually conditioned on a task representation and optimized (Zhou et al., 2019). An alternative is to use hypernetworks to generate the policy's parameters. These have been shown to make better use of task representations (Beck et al., 2023a; Beukman et al., 2024).

Exploration in Meta-RL A considerable body of literature also focuses on exploration in meta-RL. As opposed to standard RL, meta-RL exploration strategies can be learned from interactions with the environment and then applied to new tasks. Since identifying and solving RL tasks requires exploration, all meta-RL algorithms learn to explore, at least implicitly. However, several works have shown the benefits of explicitly learning to explore. Rakelly et al. (2019) use posterior sampling to explore. Zintgraf et al. (2021b) consider Bayes-optimal policies, which optimally trade-off exploration and exploitation, and meta-learn approximations of such policies. They later extend their work in Zintgraf et al. (2021c) by encouraging the agent to explore novel hyper-states during meta-training. Some other approaches use contrastive learning (Fu et al., 2021; He et al., 2024; Yu et al., 2024), information gain (Liu et al., 2021; Jiang et al., 2021; Zhang et al., 2021), or task clustering (Chu et al., 2024), to structure the exploration space, making exploration more efficient. Gradient-based meta-RL can also explicitly learn to explore. Gupta et al. (2018) explore by adding structured, meta-learned noise to the policy. Similarly, Stadie et al. (2018) enhance MAML and RL^2 by adding an exploration term to the meta-RL objective. Gurumurthy et al. (2020) add self-supervised objectives to lower variance during exploration. Finally, several of the works discussed improve exploration even further by decoupling the exploration and task-solving policies (Zhou et al., 2019; Gurumurthy et al., 2020; Liu et al., 2021; Fu et al., 2021; Norman & Clune, 2024).

5 Experiments

In this section, we present empirical results for LaSER. We first explain our meta-testing setup in Sec. 5.1. In Sec. 5.2, we demonstrate that LaSER outperforms existing meta-RL algorithms. Next, we analyze our novel approach of meta-training task policies in Sec. 5.3. Our results show that during meta-testing, our policies leverage task representations to adapt more than previous methods. Finally, we assess LaSER's task exploration and task learning abilities in Sec. 5.4 by visualizing the latent task representations computed during meta-testing.

5.1 Experimental Setup

We evaluate LaSER on the meta-RL benchmark MEWA (Stoican et al., 2023).[3] MEWA provides a task distribution in which tasks contain critical states that can lead to mistakes, which negatively affect the final return. In MEWA, there are four different types of mistakes. The probability of a mistake of each type happening depends on each particular task. The agent's goal is to avoid mistakes without delaying task completion. This can be achieved by exploring and learning each task. Importantly, Stoican et al. (2023) ensure MEWA's task distribution has no globally optimal policy (i.e. no policy can zero-shot solve all tasks). So, an agent can only be optimal by adapting, as new data is being collected. This property makes MEWA ideal for our case, as it allows us to test LaSER's ability to explore tasks.

We compare our method with three other meta-RL algorithms, MAML (Finn et al., 2017), PEARL (Rakelly et al., 2019), and VariBAD (Zintgraf et al., 2021b). We meta-train each algorithm on a truncated distribution of MEWA's overall task distribution. Given this meta-training distribution, we sample $3,000$ meta-training tasks. We then meta-test on the curated set of tasks proposed by Stoican et al. (2023), which contains 12 different tasks. These are outside the meta-training distribution. Additionally, no globally optimal policy exists for these tasks. Finally, the difference in performance between the optimal non-adaptive and optimal adaptive policies is high enough to evaluate the agent's ability to generalize and adapt. For each meta-testing task, the agents are allowed to collect $K = 4$ episodes during the task exploration phase. Then, they are evaluated on solving the task by collecting a single exploitation episode, using the task policy. Besides this final return, we also analyze an agent's ability to improve its performance as more data is collected. All tasks have a horizon of $H = 50$.

LaSER's hyperparameters are listed in Appendix H. We selected these based on performance on MEWA's entire task distribution. This includes, but is not limited to, MEWA's meta-training task distribution. Therefore, we tune for the goal of out-of-distribution generalization. For implementation details and hyperparameters for MAML, PEARL, and VariBAD, see Appendix G.

MEWA offers 3 baselines to compare our agents to. The *Random* baseline represents the expected performance of an agent that takes uniformly random actions. The *Task-Agnostic* baseline represents an agent that always takes optimal actions in MEWA's non-critical states. These are states in which the optimal action can be computed even if the policy has no task information, i.e. for the optimal task policy π^*, if

---

[3] We use the publicly available code for the MEWA benchmark at https://github.com/RStoican/MEWA.

$\pi^*(a \mid s, z) \leq \pi^*(a \mid s)$ for all z, then s is a non-critical state. For any critical state, the *Task-Agnostic* baseline considers all actions that could potentially be optimal in a valid task, then takes one at random. Finally, the *Optimal* baseline shows the expected return of the optimal meta-RL agent, i.e. the agent that adapts to and solves each task optimally.

5.2 Performance

We first evaluate LaSER's ability to solve tasks from the MEWA benchmark. We begin by showing that all algorithms converge during meta-training. For this, we evaluate agents on tasks from MEWA's meta-training distribution. As shown in Fig. 4, LaSER's task policy, encoder, and exploration policy converge across 5 seeds. We present the metrics in Fig. 4 using an exponential moving average (EMA). Specifically, each point x_i is smoothed as $s_i = \alpha_{\text{EMA}} x_i + (1 - \alpha_{\text{EMA}}) s_{i-1}$, where α_{EMA} is a constant. The encoder loss is computed using the loss function in Eq. 9. Additionally, Fig. 8 shows convergence results for each term in Eq. 9. The exploration return is computed using intrinsic rewards (see Eq. 4) collected by the exploration policy $\pi_\phi^{\text{explore}}$. Since $\pi_\phi^{\text{explore}}$ is conditioned on representations learned by the encoder, we first train the encoder for 50 million timesteps without updating $\pi_\phi^{\text{explore}}$. For these initial updates, we use data collected by a uniformly random policy. Finally, the task policies of all other algorithms also converge. The same holds for PEARL's and VariBAD's encoders (see Fig. 9). Note that MAML does not have an encoder, and LaSER is the only algorithm that uses an exploration policy.

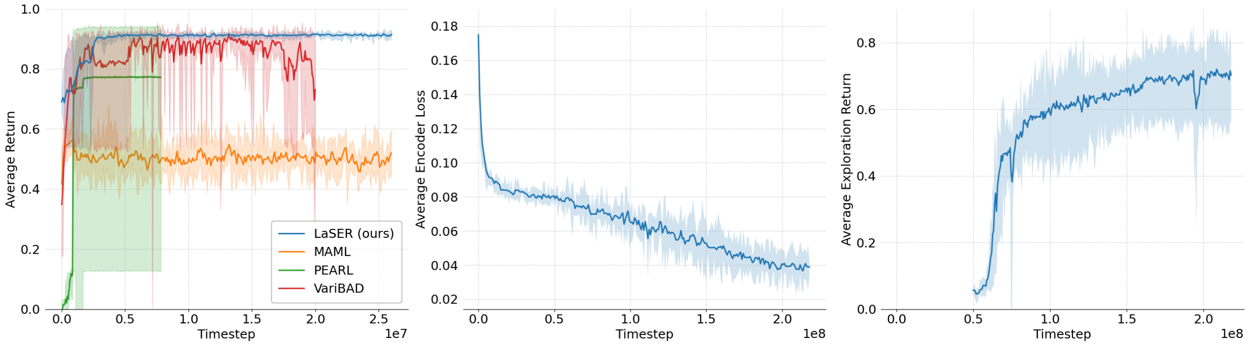


Figure 4: Performance on tasks from MEWA's meta-training task distribution. Each metric is averaged over 5 seeds. We use EMA smoothing with $\alpha_{\text{EMA}} = 0.6$. The shaded areas represent the minimum and maximum across all seeds. (left) Undiscounted return of policy π_θ. (middle) Loss $\mathcal{L}_{\text{LaSER}}$ of LaSER's encoder. (right) Undiscounted exploration return of policy $\pi_\phi^{\text{explore}}$.

For each algorithm, we meta-test the agent obtained at the end of meta-training. We stop VariBAD early, after 20 million timesteps, due to performance degradation. Similarly, we stopped PEARL after approximately 8 million timesteps, as its performance does not improve anymore.

The meta-trained agents' average returns over 5 seeds are shown in Fig. 5. For each seed, we run the agents 180 times on each of the 12 meta-testing tasks. We then report the average return over all these runs. LaSER outperforms all other meta-RL algorithms in terms of average return. It is also the only one to outperform the *Task-Agnostic* baseline. All other algorithms successfully outperform the *Random* baseline. PEARL learns a strong global policy but fails to outperform the *Task-Agnostic* baseline. In contrast, both MAML and VariBAD fail to learn a policy that reaches this baseline.

Furthermore, LaSER proves to be the most stable algorithm, with a post-adaptation standard deviation of just 0.016 across all seeds. In contrast, the other algorithms exhibit a higher variability: 0.095 for MAML, 0.441 for PEARL, and 0.262 for VariBAD.

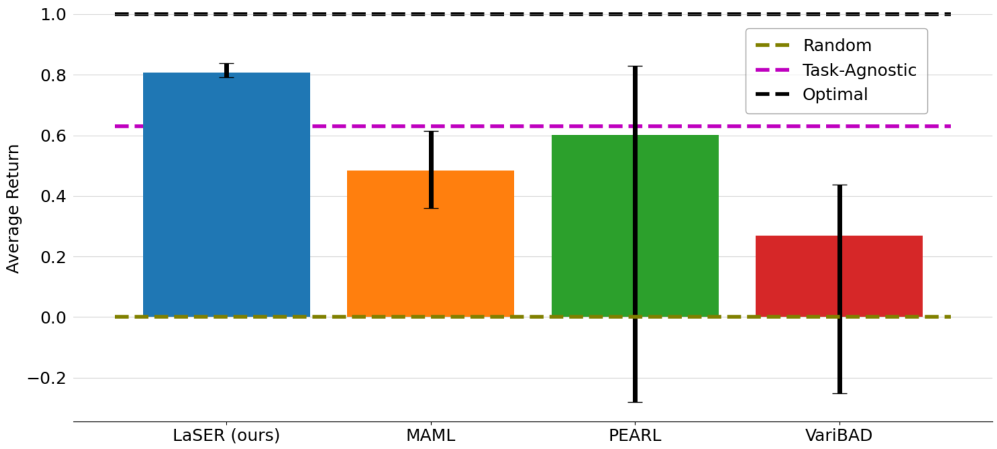


Figure 5: Returns achieved on MEWA's meta-testing tasks, averaged over 5 seeds. Each agent is meta-tested with an exploration budget of $K = 4$. Error bars indicate the minimum and maximum return across seeds. Note that we normalize results between the *Random* and *Optimal* baselines.

5.3 Adaptability

We evaluate our approach for meta-training task policies, introduced in Sec. 3.1. We meta-test task policies by rolling them out both before and after they receive a task descriptor z. During meta-testing, we roll out task policies both before and after receiving a task descriptor z. Our primary objective is to assess adaptability, i.e. a policy's ability to use z to improve its performance. We quantify this as the difference between post-adaptation and pre-adaptation performance.

To ensure a fair comparison of task policies alone, we meta-train and meta-test without task exploration or task learning. For each task, we use a ground-truth, oracle-provided, task descriptor z, which is normally unavailable to the agent. Additionally, to assess adaptability to new information, we use a simpler, multi-task objective. We run a constrained version of MEWA, meta-training and meta-testing on the same set of 12 tasks. Despite this restriction, MEWA'a guarantee of no globally optimal policy still holds.

Our method combines PPO and the proposed hyper-reward r_t^+ to optimize π_θ. We compare it against the simpler approach of using standard PPO, and against task-conditioned hypernetworks, as proposed by Beukman et al. (2024). We provide implementation details and hyperparameters for these hypernetwork policies in Appendix G.4.

The meta-training results are shown in Fig. 6. Our method achieves the highest average return. It is also the only one to reach the optimal policy, which occurred in 2/5 seeds. Our hyper-reward also stabilizes meta-training. While hypernetworks can potentially outperform standard PPO, they are highly unstable. The instability of standard PPO is presumably caused by the policy's inability to maximize returns across all tasks. Our approach also outperforms in terms of adaptability. After the extended exploration phase (around halfway through meta-training), our agent stabilizes. The task-conditioned policy then consistently outperforms the pre-adaptation policy. Since PPO does not explicitly optimize for adaptation, its adaptability is nearly four times lower than our method. Hypernetwork-generated policies can adapt better than PPO-trained policies but are also more unstable. Additionally, their lower average return limits the benefits of this adaptability.

Our method learns more slowly initially. This is due to additional exploration, where agents learn the state space in the context of the task space. We argue that this exploration phase is vital for the policy's future performance. While hypernetworks can potentially outperform standard PPO, we hypothesize they cannot surpass our method because they lack explicit exploration of task context. Their task-specific policies optimize a standard RL objective, so they explore accordingly.

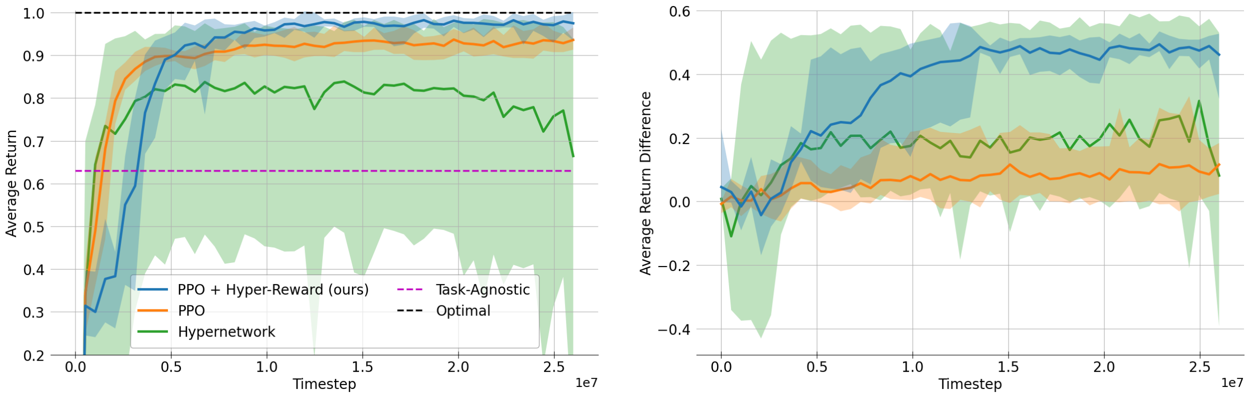


Figure 6: Task policy meta-training. We use oracle-provided task descriptors z, and meta-train and meta-test on the same set of 12 tasks. Results are averaged over 5 seeds. The shaded areas represent the maximum and minimum across seeds. (left) Return of the task policy π_θ, given a task descriptor z. All returns are normalized between MEWA's *Random* and *Optimal* baselines. To improve readability, we restrict the plot to the $[0.2, 1.0]$ range and exclude the *Random* baseline (which is positioned at 0). (right) Adaptability, quantified as the difference in return achieved by π_θ with and without access to z.

Finally, we observe that our method is nearly as computationally efficient as standard PPO and much faster than hypernetworks. We measure the average wall clock time of rolling out and optimizing π_θ. On average, hypernetwork agents are approximately 3.3× slower than our method, which is in turn only 1.3× slower than standard PPO. See Fig. 11 for a more detailed comparison. We discuss what causes the additional overhead in Appendix A.2.

5.4 Exploring and Learning Tasks

To better understand LaSER's ability to explore and encode tasks during meta-testing, we visualize its latent task space. We compare the latent task representations computed by LaSER, PEARL, and VariBAD for each of the 12 meta-testing tasks. For each algorithm, we show results for only one of the meta-training seeds. However, we obtain similar representations for the other seeds. We first roll out the corresponding agent's meta-trained policy, collecting 100 meta-episodes per task, with $K = 4$ episodes per meta-episode. Note that in the case of LaSER, we roll out the exploration policy $\pi_\phi^{\text{explore}}$. We then encode each meta-episode $\mathcal{D}^{(K)}$ into a latent representation z using the agent's meta-trained encoder. We visualize two-dimensional projections of z, computed using t-SNE (Van der Maaten & Hinton, 2008), in Fig. 7. The 12 tasks are sorted by similarity, i.e. average mistake probability, as in Table 1.

For LaSER, we can find clusters of task representations. Additionally, we can see meta-episodes collected from the first five tasks being projected apart from those collected from the last seven tasks. Note that in the former group, mistakes of types III or IV cannot happen, while in the latter they can (see Table 1). The separation seems to follow this trend. In contrast, PEARL's and VariBAD's latent representations lack clustering that could distinguish data collected from different tasks. This low separation could make it more difficult to identify task-specific features.

We argue that learning higher-quality clusters is not trivial. The difficulty may come from the high variance in episodes collected from the most difficult tasks. These tasks limit the exploration policy's control over the states it encounters. Additionally, the low adaptation budget K can make it difficult to separate representations of similar tasks.

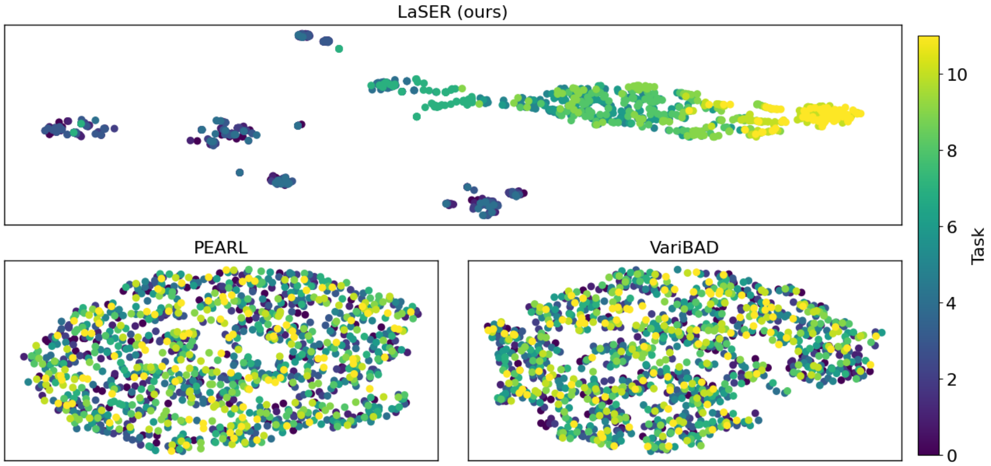


Figure 7: Latent representations of meta-episodes collected and encoded by meta-trained LaSER, PEARL, and VariBAD agents. Each of the 1200 points is a latent representation belonging to one of the 12 meta-testing tasks. Different tasks have different colors.

6 Conclusion

We introduced LaSER, a new approach for meta-learning RL exploration. Our results demonstrated that LaSER outperforms previous meta-RL algorithms on the MEWA benchmark. They also showed that LaSER can meta-learn better task clustering during exploration. Additionally, we proposed a novel method for meta-training task policies efficiently. By using a meta-learned exploration bonus, we outperform previous approaches in both performance and adaptability. Since our approach is agnostic to how tasks are explored or represented, it can be integrated into any context-based meta-RL algorithm. Then, it can be optimized using standard RL methods.

An important strength of LaSER is that its pre-adaptation policy already outperforms the other algorithms. We argue that this is a result of meta-training with higher-quality task representations. The main role of these representations is to provide task information for adaptation, during meta-testing. However, our results suggest they may also enhance performance and sample efficiency during meta-training. We believe this secondary role should also be investigated.

LaSER is built upon the unique properties and requirements of the meta-RL framework. We approach this framework from the perspective of few-shot adaptation. LaSER's exploration algorithm leverages key assumptions about the data collected in few-shot adaptation environments. Similarly, at the center of LaSER's task policy meta-training regime stands an important inductive bias. Incorporating the expectation that meta-RL policies outperform standard RL policies in a meta-RL setting is enough for adaptive behavior to emerge. We hope our findings inspire future work to leverage the meta-RL framework in novel ways.

While LaSER adapts better than previous methods in several MEWA tasks, a drawback is its low average adaptability across all tasks. We discuss this from the perspective of LaSER's three main components: exploration policy, encoder, and task policy. Empirical results suggest that the first two components are well-optimized for exploring and learning tasks effectively. For example, the encoder learned to separate MEWA's first 5 meta-testing tasks from the rest. This suggests that LaSER should be able to outperform its pre-adaptation policy in this type of task, i.e. tasks with low average mistake probability. However, it instead chooses to not attempt adaptation. We further show that LaSER's task policy becomes almost optimally adaptive when provided with appropriate task representations. Despite this, effectively combining these three components appears to be non-trivial. We therefore suggest that future research should seek to explore and understand this issue.

Acknowledgments

Acknowledgments will be added once the paper is accepted.

References

Karol Arndt, Murtaza Hazara, Ali Ghadirzadeh, and Ville Kyrki. Meta reinforcement learning for sim-to-real domain adaptation. In *2020 IEEE international conference on robotics and automation (ICRA)*, pp. 2725–2731. IEEE, 2020.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Anand Ballou, Xavier Alameda-Pineda, and Chris Reinke. Variational meta reinforcement learning for social robotics. *Applied Intelligence*, 53(22):27249–27268, 2023.

Andrea Banino, Adrià Puigdomenech Badia, Jacob C Walker, Tim Scholtes, Jovana Mitrovic, and Charles Blundell. Coberl: Contrastive bert for reinforcement learning. In *International Conference on Learning Representations, ICLR*, 2015.

Jacob Beck, Matthew Thomas Jackson, Risto Vuorio, and Shimon Whiteson. Hypernetworks in meta-reinforcement learning. In *Conference on Robot Learning*, pp. 1478–1487. PMLR, 2023a.

Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023b.

Jacob Beck, Matthew Thomas Jackson, Risto Vuorio, Zheng Xiong, and Shimon Whiteson. SplAgger: Split aggregation for meta-reinforcement learning. *Reinforcement Learning Journal*, 1:450–469, 2024.

Michael Beukman, Devon Jarvis, Richard Klein, Steven James, and Benjamin Rosman. Dynamics generalisation in reinforcement learning via adaptive context-aware policies. *Advances in Neural Information Processing Systems*, 36, 2024.

Zhenshan Bing, Alexander Koch, Xiangtong Yao, Kai Huang, and Alois Knoll. Meta-reinforcement learning via language instructions. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5985–5991. IEEE, 2023.

Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

Zhendong Chu, Renqin Cai, and Hongning Wang. Meta-reinforcement learning via exploratory task clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 11633–11641, 2024.

Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. In *Conference on Robot Learning*, pp. 617–629. PMLR, 2018.

Tristan Deleu. Model-Agnostic Meta-Learning for Reinforcement Learning in PyTorch, 2018. Available at: https://github.com/tristandeleu/pytorch-maml-rl.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, 2019.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Haotian Fu, Hongyao Tang, Jianye Hao, Chen Chen, Xidong Feng, Dong Li, and Wulong Liu. Towards effective context for meta-reinforcement learning: an approach based on contrastive learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35(8), pp. 7457–7465, 2021.

Yuan Gao, Elena Sibirtseva, Ginevra Castellano, and Danica Kragic. Fast adaptation with meta-reinforcement learning for trust modelling in human-robot interaction. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 305–312. IEEE, 2019.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. *Advances in neural information processing systems*, 31, 2018.

Swaminathan Gurumurthy, Sumit Kumar, and Katia Sycara. Mame: Model-agnostic meta-exploration. In *Conference on Robot Learning*, pp. 910–922. PMLR, 2020.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *5th International Conference on Learning Representations*, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Hongcai He, Anjie Zhu, Shuang Liang, Feiyu Chen, and Jie Shao. Decoupling meta-reinforcement learning with gaussian task contexts and skills. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38(11), pp. 12358–12366, 2024.

Jerry Zhi-Yang He, Zackory Erickson, Daniel S Brown, Aditi Raghunathan, and Anca Dragan. Learning representations that enable generalization in assistive tasks. In *Conference on Robot Learning*, pp. 2105–2114. PMLR, 2023.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In *International Conference on Machine Learning*, pp. 4475–4483. PMLR, 2020.

Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for pomdps. In *International conference on machine learning*, pp. 2117–2126. PMLR, 2018.

Takahisa Imagawa, Takuya Hiraoka, and Yoshimasa Tsuruoka. Off-policy meta-reinforcement learning with belief-based task inference. *IEEE Access*, 10:49494–49507, 2022.

Peng Jiang, Shiji Song, and Gao Huang. Exploration with task information for meta reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 34(8):4033–4046, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations ICLR*, 2015.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdel rahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Annual Meeting of the Association for Computational Linguistics*, 2019.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

Evan Z Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices. In *International conference on machine learning*, pp. 6925–6935. PMLR, 2021.

Luckeciano C Melo. Transformers are meta-reinforcement learners. In *international conference on machine learning*, pp. 15340–15359. PMLR, 2022.

Reuth Mirsky, Ignacio Carlucho, Arrasy Rahman, Elliot Fosong, William Macke, Mohan Sridharan, Peter Stone, and Stefano V Albrecht. A survey of ad hoc teamwork research. In *European conference on multi-agent systems*, pp. 275–293. Springer, 2022.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *6th International Conference on Learning Representations ICLR*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Skander Moalla, Andrea Miele, Daniil Pyatko, Razvan Pascanu, and Caglar Gulcehre. No representation, no trust: Connecting representation, collapse, and trust issues in ppo. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 69652–69699. Curran Associates, Inc., 2024.

Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations*, 2018.

Ben Norman and Jeff Clune. First-explore, then exploit: Meta-learning to solve hard exploration-exploitation trade-offs. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

Georgios Papoudakis, Filippos Christianos, and Stefano Albrecht. Agent modelling under partial observability for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 34:19210–19222, 2021.

Mary Phuong and Marcus Hutter. Formal algorithms for transformers. *arXiv preprint arXiv:2207.09238*, 2022.

Vihari Piratla, Praneeth Netrapalli, and Sunita Sarawagi. Efficient domain generalization via common-specific low-rank decomposition. In *International Conference on Machine Learning*, pp. 7728–7738. PMLR, 2020.

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pp. 5331–5340. PMLR, 2019.

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Gresa Shala, André Biedenkapp, and Josif Grabocka. Hierarchical transformers are efficient meta-reinforcement learners. *arXiv preprint arXiv:2402.06402*, 2024.

Bradly C Stadie, Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel, and Ilya Sutskever. Some considerations on learning to explore via meta-reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

Radu Stoican, Angelo Cangelosi, and Thomas H Weisswange. Mewa: A benchmark for meta-learning in collaborative working agents. In *2023 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1435–1442. IEEE, 2023.

Mingfei Sun, Vitaly Kurin, Guoqing Liu, Sam Devlin, Tao Qin, Katja Hofmann, and Shimon Whiteson. You may not need ratio clipping in ppo. *arXiv preprint arXiv:2202.00079*, 2022.

Mingfei Sun, Sam Devlin, Jacob Beck, Katja Hofmann, and Shimon Whiteson. Trust region bounds for decentralized ppo under non-stationarity. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pp. 5–13, 2023.

Flood Sung, Li Zhang, Tao Xiang, Timothy Hospedales, and Yongxin Yang. Learning to learn: Meta-critic networks for sample efficient learning. *arXiv preprint arXiv:1706.09529*, 2017.

Richard S Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.

Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pp. 3–17. Springer, 1998.

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

M Alex O Vasilescu. *A Multilinear (Tensor) Algebraic Framework for Computer Graphics, Computer Vision and Machine Learning*. PhD thesis, University of Toronto, 2009.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Haotian Xu, Qi Fang, Cong Hu, Yue Hu, and Quanjun Yin. Mira: Model-based imagined rollouts augmentation for non-stationarity in multi-agent systems. *Mathematics*, 10(17):3059, 2022.

Jiachen Yang, Ethan Wang, Rakshit Trivedi, Tuo Zhao, and Hongyuan Zha. Adaptive incentive design with multi-agent meta-gradient reinforcement learning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pp. 1436–1445, 2022.

Xuehui Yu, Mhairi Dunion, Xin Li, and Stefano V Albrecht. Skill-aware mutual information optimisation for zero-shot generalisation in reinforcement learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Jin Zhang, Jianhao Wang, Hao Hu, Tong Chen, Yingfeng Chen, Changjie Fan, and Chongjie Zhang. Metacure: Meta reinforcement learning with empowerment-driven exploration. In *International Conference on Machine Learning*, pp. 12600–12610. PMLR, 2021.

Tony Z Zhao, Jianlan Luo, Oleg Sushkov, Rugile Pevceviciute, Nicolas Heess, Jon Scholz, Stefan Schaal, and Sergey Levine. Offline meta-reinforcement learning for industrial insertion. In *2022 international conference on robotics and automation (ICRA)*, pp. 6386–6393. IEEE, 2022.

Wenxuan Zhou, Lerrel Pinto, and Abhinav Gupta. Environment probing interaction policies. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pp. 7693–7702. PMLR, 2019.

Luisa Zintgraf, Sam Devlin, Kamil Ciosek, Shimon Whiteson, and Katja Hofmann. Deep interactive bayesian reinforcement learning via meta-learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1712–1714, 2021a.

Luisa Zintgraf, Sebastian Schulze, Cong Lu, Leo Feng, Maximilian Igl, Kyriacos Shiarlis, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: variational bayes-adaptive deep rl via meta-learning. *The Journal of Machine Learning Research*, 22(1):13198–13236, 2021b.

Luisa M Zintgraf, Leo Feng, Cong Lu, Maximilian Igl, Kristian Hartikainen, Katja Hofmann, and Shimon Whiteson. Exploration in approximate hyper-state space for meta reinforcement learning. In *International Conference on Machine Learning*, pp. 12991–13001. PMLR, 2021c.

A Policy Optimization in Meta-RL

LaSER uses the proximal policy optimization (PPO) algorithm (Schulman et al., 2017) to optimize both its task-exploration and task-solving policies. Therefore, we provide a short technical description of PPO in Appendix A.1. Then, in Appendix A.2, we show a detailed overview of how PPO can be used in meta-RL, by implementing the method proposed in Sec. 3.1.

A.1 PPO Background

For a policy π_θ with parameters θ, Schulman et al. (2017) propose the objective

$$\mathcal{L}_t^{\text{PPO}}(\theta) = \mathbb{E}_t\left[\mathcal{L}_t^{\text{CLIP}}(\theta) - c_1\mathcal{L}_t^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t, z)\right], \tag{10}$$

where c_1, c_2 are constants, $\mathcal{L}_t^{\text{CLIP}}$ is the main PPO objective, and $\mathcal{L}_t^{\text{VF}}$ and $S[\pi_\theta]$ are additional objectives. $\mathcal{L}_t^{\text{VF}}$ is a squared-error loss on the value function $V_\theta(s_t, z)$, while $S[\pi_\theta]$ is the policy entropy. Note that the sole change from the original description of PPO is that $\pi_\theta(a_t \mid s_t, z)$ and $V_\theta(s_t, z)$ depend not only on the state s_t, but also on the task descriptor z. The clipped surrogate objective $\mathcal{L}_t^{\text{CLIP}}$ is defined as

$$\mathcal{L}_t^{\text{CLIP}} = \mathbb{E}_t\Big[\min\Big(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\Big)\Big], \tag{11}$$

for a constant ϵ, policy probability ratio $r_t(\theta)$, and estimated advantage $\hat{A}_t$. The probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t,z)}{\pi_{\theta_{\text{old}}}(a_t|s_t,z)}$ is computed using the parameters θ_{old} from before an update. Eq. 11 encourages small, stable policy updates that keep π_θ close to $\pi_{\theta_{\text{old}}}$, by constraining $r_t(\theta)$ to remain within $[1-\epsilon, 1+\epsilon]$.

A popular choice for the advantage estimator $\hat{A}_t$ is the generalized advantage estimator (GAE) (Schulman et al., 2015). In the meta-RL setting, the GAE can be defined as $\hat{A}_t^{\text{GAE}} = \sum_{l=0}^{H-t}(\gamma\lambda)^l \delta_{t+l}^V$ for an episode τ with horizon H. Here, $\gamma \in [0, 1]$ is the MDP's discount factor, $\lambda \in [0, 1]$ is an additional discount, and $\delta_t^V = r_t + \gamma V(s_{t+1}, z) - V(s_t, z)$ is the temporal-difference (TD) error at timestep t for a reward r_t.

A.2 PPO for Meta-RL

In Sec. 3.1, we propose a simple change to the policy optimization objective. By replacing the environment reward r_t with our proposed hyper-reward r_t^+, PPO can be used to optimize π_θ to solve meta-RL tasks. That is, we compute the TD error

$$\delta_t^V(\theta) = r_t + \beta\, w(s_t, z)\, S[\pi_\theta](s_t, z) + \gamma V(s_{t+1}, z) - V(s_t, z), \tag{12}$$

where $w(s_t, z) = \max(0, \tanh(V(s_t, \cdot) - V(s_t, z) - \zeta))$. The estimated advantages can then be computed as $\hat{A}_t^{\text{GAE}}(\theta) = \sum_{l=0}^{H-t}(\gamma\lambda)^l \delta_{t+l}^V(\theta)$. Note that, because $\hat{A}_t^{\text{GAE}}(\theta)$ is now a function of the parameters θ, the advantages must be recomputed every time θ is updated, i.e. after each PPO minibatch update. This is in contrast to standard PPO, where $\hat{A}_t^{\text{GAE}}$ is computed only once, before an update, and then kept fixed until new data is collected.

Intuitively, $w(s_t, z)$ measures how effective the policy π_θ is at using the context z, for each timestep t. The assumption is that, while PPO-optimized policies in single-MDP settings might explore the state space sufficiently, there is no guarantee that the task descriptor space is explored enough in a meta-RL setting. If z does not lead to an improvement above a threshold ζ, the policy is urged to explore from state s_t, thus learning more about z. As the agent improves at using z to maximize return, the exploration bonus $S[\pi_\theta]$ for state s_t decreases. The standard PPO objective is only recovered when $V(s_t, \cdot) \leq V(s_t, z) + \zeta$. This signals that the agent has learned how to use z in state s_t, and no additional exploration is required.

B Masked Self-Supervised Training

The bidirectional transformer encoder f_b, introduced in Sec. 3, learns useful data representations by learning to reconstruct its input. Since reconstructing a meta-episode $\mathbf{B}_j$ is trivial when the entire $\mathbf{B}_j$ is given as input, we use masked self-supervised training. We create a masked meta-episode $\mathbf{B}_j^{\text{masked}}$ by applying a stochastic masking function, similarly to Devlin et al. (2019); Lewis et al. (2019). More precisely, $\mathbf{B}_j^{\text{masked}}$ is identical to $\mathbf{B}_j$, with the exception that each timestep (s, a, r) in $\mathbf{B}_j^{\text{masked}}$ has a 15% chance of being corrupted. The loss $\mathcal{L}_{\text{rec}}$ measures the ability of f_b to predict the true value of each corrupted timestep. The encoder must learn to compute these values from the non-corrupted timesteps in the input. We consider two types of corruption.

- **Masking**: with a probability of 80%, the selected timestep is replaced by a special $\langle MASK \rangle$ timestep, which carries no information.
- **Replacing**: with a probability of 10%, the selected timestep is replaced by another timestep from the same meta-episode.

The rest (i.e. 10%) of the selected timesteps are not corrupted but are still used when computing $\mathcal{L}_{\text{rec}}$. To optimize this loss, f_b must learn general temporal relationships between the timesteps in a meta-episode. We compute $\mathcal{L}_{\text{rec}}$ as the MSE between the unmasked tokens in $\mathbf{B}_j$ and the reconstructed meta-episode $\mathbf{B}'_j = h_{\text{rec}}(z; \omega_{\text{rec}})$, where $z = g(\mathbf{B}_j^{\text{masked}}; \omega)$. Note that masked meta-episodes are only used to train the encoder. When training the task policy π_θ or during meta-testing, we compute $z = g(\mathbf{B}_j; \omega)$ using unmasked meta-episodes.

C Algorithms

For the sake of completeness, but also to enable reproducibility and further analysis (Phuong & Hutter, 2022), we provide pseudocode for our proposed algorithm LaSER. Alg. 1 shows our meta-training process. It is composed of three main parts. Each part optimizes one of the three LaSER components: the encoder g_ω (Alg. 3), the exploration policy $\pi_\phi^{\text{explore}}$ (Alg. 4), and the task policy π_θ (Alg. 5). Finally, we provide details on how we meta-test a fully trained LaSER agent in Alg. 2.

D Meta-Training and Meta-Testing Tasks

We provide more details on the tasks available in the MEWA benchmark. MEWA evaluates agents on their ability to take optimal actions in "critical states". These critical states provide agents with two options. Consider a critical state s_x of type x. An agent's first option is to take a risky action a_{risk}, potentially reaching a state s' and making progress in the task. However, action a_{risk} could also lead to a state s'_x. This state denotes that a mistake has happened, leading to a large delay in task completion and resulting

Algorithm 1 LaSER Meta-Training

```
    **Input** $p(\mathcal{M})$, task distribution
    **Output** $\pi_\theta$, task policy; $\pi_\phi^{\text{explore}}$, exploration policy; $g_\omega$, encoder; $\hat{\boldsymbol{z}}_c$, shared component
$\theta, \phi, \omega \leftarrow$ initialize randomly
**for** $n = 1, 2, \ldots, N_{\text{explore}}$ **do**
    $\omega \leftarrow$ train_encoder($p(\mathcal{M}), \pi_\phi^{\text{explore}}, g_\omega$)
    $\phi \leftarrow$ train_exploration_policy ($p(\mathcal{M}), \pi_\phi^{\text{explore}}, g_\omega$)
**end for**
$\mathcal{B} \leftarrow$ [sample $Q$ meta-episodes $\mathbf{B} \sim P_{\mathcal{M}_i}^{\pi_\phi^{\text{explore}}}$ **for** $\mathcal{M}_i \sim p(\mathcal{M})$]
$\hat{\boldsymbol{z}}_c, \_ \leftarrow f_b(\mathcal{B}; \omega_b)$
**for** $n = 1, 2, \ldots, N_{\text{task}}$ **do**
    $\theta \leftarrow$ train_task_policy($p(\mathcal{M}), \pi_\theta, \pi_\phi^{\text{explore}}, g_\omega, \boldsymbol{z}_c$)
**end for**
**return** $\pi_\theta, \pi_\phi^{\text{explore}}, g_\omega, \hat{\boldsymbol{z}}_c$
```

Algorithm 2 LaSER Meta-Testing

```
    **Input** $p(\mathcal{M})$, task distribution; $\pi_\theta$, task policy; $\pi_\phi^{\text{explore}}$, exploration policy; $g_\omega$, encoder;
            $\hat{\boldsymbol{z}}_c$, shared component
**for** $\mathcal{M}_i \sim p(\mathcal{M})$ **do**
    Sample exploration meta-episode $\mathbf{B}_j \sim P_{\mathcal{M}_i}^{\pi_\phi^{\text{explore}}}$
    $z \leftarrow g(\mathbf{B}_j; \omega)$, using $\hat{\boldsymbol{z}}_c$ as the shared component
    $\tau \sim P_{\mathcal{M}_i}^{\pi_\theta(a|s,z)}$
    Measure return in exploitation episode $\tau$
**end for**
```

Algorithm 3 train_encoder()

```
    **Input** $p(\mathcal{M})$, task distribution; $\pi_\phi^{\text{explore}}$, exploration policy; $g_\omega$, encoder
    **Output** $\omega$, updated parameters
$\mathcal{B} \leftarrow$ [sample $Q$ meta-episodes $\mathbf{B} \sim P_{\mathcal{M}_i}^{\pi_\phi^{\text{explore}}}$ **for** $\mathcal{M}_i \sim p(\mathcal{M})$]
$\mathcal{B}^{\text{masked}} \leftarrow$ mask($\mathcal{B}$)
**for** $j = 1, 2, \ldots, N_{\text{encoder}}$ **do**
    Compute $z$ as $g(\mathbf{B}^{\text{masked}}; \omega)$ for each $\mathbf{B}^{\text{masked}} \in \mathcal{B}^{\text{masked}}$
    Compute $\mathbf{C}$ using $\omega_{\text{t-rec}}$ and $\delta$ using Eq. 7 for each $\mathbf{B}^{\text{masked}} \in \mathcal{B}^{\text{masked}}$
    Compute the reconstructed input $\mathcal{B}'$ using $\omega_{\text{rec}}$
    $\mathcal{L}_{\text{LaSER}}(\omega, \omega_{\text{rec}}, \omega_{\text{t-rec}}) \leftarrow$ compute using Eq. 6, 8, 9
    $\omega, \omega_{\text{rec}}, \omega_{\text{t-rec}} \leftarrow$ update using $\nabla \mathcal{L}_{\text{LaSER}}$
**end for**
**return** $\omega$
```

Algorithm 4 train_exploration_policy()

```
**Input** $p(\mathcal{M})$, task distribution; $\pi_\phi^{\text{explore}}$, exploration policy; $g_\omega$, encoder
**Output** $\phi$, updated parameters
$\mathcal{D} \leftarrow []$
for $\mathcal{M}_i \sim p(\mathcal{M})$ do
    for $k = 1, 2, \dots, K$ do
        $\mathcal{D}^{(K)} \leftarrow []$
        for $t = 1, 2, \dots, H$ do
            $a_t^k \sim \pi_\phi^{\text{explore}}\left(\cdot \mid s_t^k, \Gamma^{(:k,:t-1)}\right)$
            $\mathcal{D}^{(:k,:t)} \leftarrow (s_t^k, a_t^k, r_t^k)$
            $\Gamma^{(:k,:t)} \leftarrow f_u\left(\mathcal{D}^{(:k,:t)};\ \omega_u\right)$
            Collect $s_{t+1}^k, r_{t+1}^k$ by taking action $a_t^k$ in $\mathcal{M}_i$
        end for
    end for
    $\Gamma \leftarrow f_u(\mathcal{D}^{(K)};\ \omega_u)$;        $\boldsymbol{d} \leftarrow \frac{1}{K}(\Gamma^T\Gamma)^T\mathbf{1}$
    Compute $\tilde{R}_k(s_t^k, a_t^k)$ using Eq. 4 for each $k, t$
    Replace environment rewards in $\mathcal{D}^{(K)}$ with the corresponding $\tilde{R}_k(s_t, a_t)$
    $\mathcal{D} \leftarrow [\mathcal{D}, \mathcal{D}^{(K)}]$
end for
for $j = 1, 2, \dots, N_{\text{PPO}}$ do
    Optimize $\phi$ on transitions from $\mathcal{D}$ using PPO
end for
return $\phi$
```

Algorithm 5 train_task_policy()

```
**Input** $p(\mathcal{M})$, task distribution; $\pi_\theta$, task policy; $\pi_\phi^{\text{explore}}$, exploration policy; $g_\omega$, encoder;
        $\hat{\boldsymbol{z}}_c$, shared component
**Output** $\theta$, updated parameters
$\mathcal{D} \leftarrow []$
for $\mathcal{M}_i \sim p(\mathcal{M})$ do
    Sample exploration meta-episode $\mathbf{B}_j \sim P_{\mathcal{M}_i}^{\pi_\phi^{\text{explore}}}$
    $z \leftarrow g(\mathbf{B}_j;\ \omega)$, using $\hat{\boldsymbol{z}}_c$ as the shared component
    $\mathcal{D} \leftarrow [\mathcal{D},\ \tau \sim P_{\mathcal{M}_i}^{\pi_\theta(a|s,z)}]$
end for
for $j = 1, 2, \dots, N_{\text{PPO}}$ do
    for $\tau \in \mathcal{D}$ do
        for $t = 1, 2, \dots, H$ do
            Compute $V(s_t, z),\ V(s_t, \cdot)$, and $S[\pi_\theta](s_t, z)$
            $w(s_t, z) \leftarrow$ compute using Eq. 3
            $r_t^+ \leftarrow r_t + \beta w(s_t, z) S[\pi_\theta](s_t, z)$
            Compute and store $\hat{A}_t^{\text{GAE}}(\theta)$ using $r_t^+$
        end for
    end for
    Optimize $\theta$ on transitions from $\mathcal{D}$ using PPO
end for
return $\theta$
```

in lower returns. The probability of s'_x being reached depends on both x and the task. The agent's second option is to take a safe action a_{safe}. This provides a small delay and follows the transition probabilities $T_i(s_{x-1} \mid s_x, a_{\text{safe}}, \mathcal{M}_i) = 1$. That is, the agent reaches a critical state s_{x-1} of type $x - 1$. An important feature of MEWA's task distribution is that for any task and any $y \leq x$, mistakes of type y are less likely than mistakes of type x. Therefore, depending on the task, taking several safe actions before a risky action could be optimal.

In this work, all agents are meta-trained on the narrow task distribution analyzed by Stoican et al. (2023). This corresponds to a task distribution $p(\mathcal{M})$ in which any task $\mathcal{M}_i \sim p(\mathcal{M})$ has four different types of critical states. That is, $\mathcal{M}_i$ can be described by a vector of probabilities $p_i \in \mathbb{R}^4$. The probability of a mistake of type $x \in \{1, 2, 3, 4\}$ happening in $\mathcal{M}_i$ is $p_{i,x}$. More formally, let s'_x be a post-mistake state, for a mistake of type x. Then, the dynamics of $\mathcal{M}_i$ are described by $T_i(s'_x \mid s, a, \mathcal{M}_i) = p_{i,x}$, for some critical state s_x and risky action a_{risk}.

For meta-testing, we use a fixed set of 12 tasks. These are used for the results presented in Sec. 5 and in this section. Table 1 shows these meta-testing tasks, described by their corresponding vector p_i. Additionally, for each task, we provide the average probability of a mistake of any type happening. This is simply computed as $1/4 \sum_{x=1}^{4} p_{i,x}$. We use this as a measure of similarity, with similar tasks requiring similar (but not identical) optimal policies.

Task Index	**Mistake Probability**				**Average Mistake Probability**
	Type I	**Type II**	**Type III**	**Type IV**	
0	0	0	0	0	0.000
1	0.05	0	0	0	0.013
2	0.1	0	0	0	0.025
3	0.772	0	0	0	0.193
4	0.672	0.618	0	0	0.322
5	0.622	0.618	0.577	0.434	0.563
6	0.622	0.618	0.577	0.534	0.588
7	0.872	0.818	0.777	0	0.617
8	0.672	0.668	0.627	0.584	0.638
9	0.972	0.968	0.927	0.384	0.813
10	0.972	0.968	0.927	0.784	0.913
11	0.972	0.968	0.927	0.884	0.938

Table 1: The configuration of the 12 tasks used for meta-testing agents. A task is described by a 4-dimensional vector. Each dimension corresponds to the probability of a mistake of the corresponding type happening. We additionally compute the average mistake probability across all types. This can be seen as a way of comparing tasks, i.e. tasks with similar average probabilities have similar optimal policies.

E Additional Results

We provide results that complement those in Sec. 5. For LaSER, the convergence of each term in its encoder's loss function (see Eq. 9) is presented in Fig. 8. The meta-training losses of PEARL's and VariBAD's encoders are shown in Fig. 9. Fig. 10 extends the meta-testing results in Sec. 5.2 by presenting performance on each of the 12 meta-testing tasks. Fig. 11 shows the wall clock time of meta-training the task policies described in Sec. 5.3.

F Implementation Details

We provide additional details on our implementation of the LaSER algorithm and our architecture.

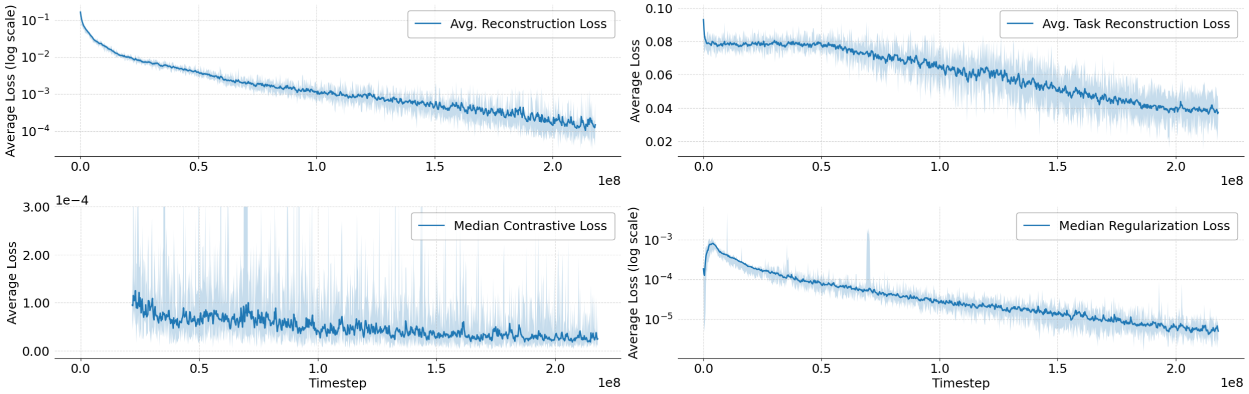


Figure 8: Encoder loss for LaSER, computed on tasks from MEWA's meta-training distribution. We show separate plots for each term in the encoder loss function. The first two metrics are averaged over 5 seeds. For the latter two, we find one of the seeds to be unstable early in meta-training, leading to high losses for approximately 5×10^5 timesteps. Therefore, we choose to provide the median over the 5 seeds. For this same reason, we restrict the plot of the contrastive loss to $[-1.5\text{e}{-5}, 3\text{e}{-4}]$. To aid readability for exponential losses, the reconstruction and regularization losses use a logarithmic scale. All metrics are smoothed using EMA with $\alpha_{\text{EMA}} = 0.4$. The minimum and maximum across seeds are given by the shaded areas.

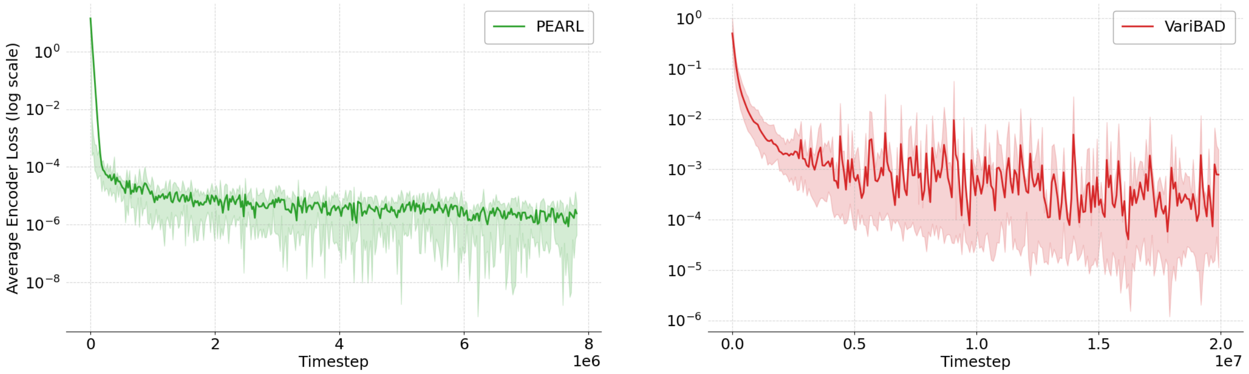


Figure 9: PEARL's and VairBAD's encoder loss on tasks from MEWA's meta-training task distribution. All metrics are averaged over 5 seeds and smoothed using EMA with $\alpha_{\text{EMA}} = 0.8$. Since the losses decay exponentially, we use logarithmic scales. The shaded areas give the minimum and maximum across seeds.

F.1 Encoder

Both f_u and f_b use the same transformer architecture (Vaswani et al., 2017), except that f_u is unidirectional, while f_b is bidirectional. Each transformer has size $d_{\text{model}} = 128$, 8 layers, 16 attention heads, and feed-forward networks of size 512, with GELU (Hendrycks & Gimpel, 2016) activation functions. Before passing the input through the transformer, we linearly map each episode timestep in d_{model}-dimensional embeddings, then add positional encodings. All input timesteps are linearly mapped to d_{model}-dimensional embeddings, with positional encodings added before they are passed through the transformer. For stable training, we apply the T-Fixup initialization scheme (Huang et al., 2020), as recommended by Melo (2022).

The output of f_u is passed through a single-layer feed-forward network that outputs Γ. Similarly, the output of f_b is independently processed by two separate single-layer feed-forward networks, outputting $\boldsymbol{z}_c$ and $\boldsymbol{Z}_s$, respectively. Each of these three networks has size 64 and uses tanh activation functions. Note that, before computing $\boldsymbol{z}_c$, average pooling is applied to reduce the dimensionality of f_b's output.

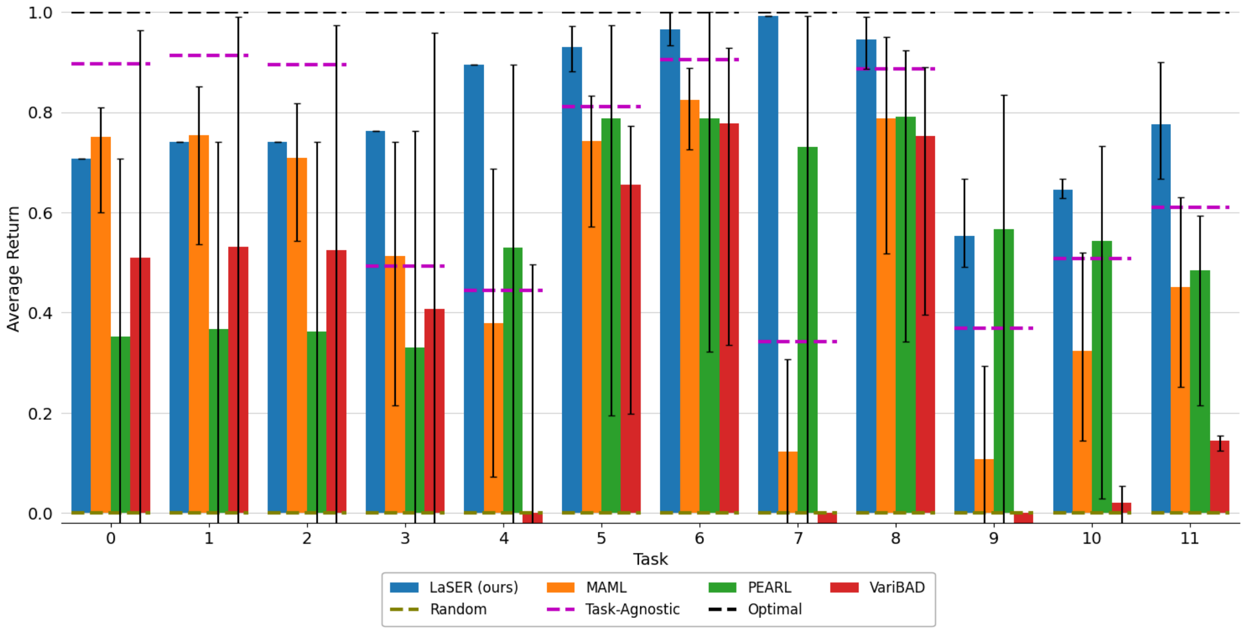


Figure 10: Meta-testing results for each of the 12 meta-testing tasks in MEWA, averaged over 5 seeds. Each agent is meta-tested with an exploration budget of $K = 4$. Error bars indicate the minimum and maximum return across seeds. The three baselines are computed on a per-task basis. We normalize returns between the *Random* and *Optimal* baselines and restrict the plot to the $[-0.02, 1.0]$ range.

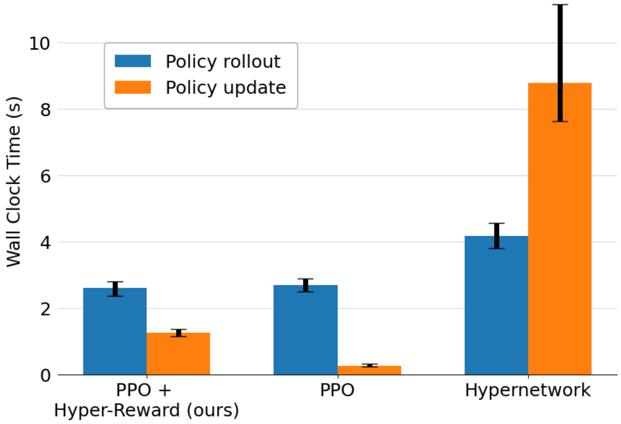


Figure 11: Wall clock time results for task policy meta-training. We measure the average time per iteration (in seconds) required to roll out episodes and optimize the policy. The results are averaged over 5 seeds. The bars represent the shortest and longest amount of time across all seeds.

These networks extract features from the transformers' output. The features are exploration-specific for Γ, task-specific for $\boldsymbol{Z}_s$, and specific to the task distribution for $\boldsymbol{z}_c$. However, the networks have a second role. They ensure Γ, $\boldsymbol{Z}_s$, and $\boldsymbol{z}_c$ are computed correctly when g_ω is updated with data from multiple tasks. Specifically, Γ and $\boldsymbol{Z}_s$ are computed for each task $\mathcal{M}_i \sim p(\mathcal{M})$ using data exclusively from that task. In contrast, $\boldsymbol{z}$ is computed with data from multiple tasks, leading to a general representation of the entire distribution.

The encoder g_ω has two final transformer layers, h, similar to those in f_b. The first layer combines $\boldsymbol{Z}_s$ and Γ. The second computes z by combining the output of the first with $\boldsymbol{z}_c$. The reconstruction head h_{rec} performs a linear transformation from d_{model} to d. The task-reconstruction head $h_{\text{t-rec}}$ is a $(256, 128, 128)$ feed-forward network with GELU activations. We use the Adam optimizer (Kingma & Ba, 2015) to train g_ω, h_{rec}, and $h_{\text{t-rec}}$.

F.2 Policies

We represent $\pi_\phi^{\text{explore}}$, V_ϕ, π_θ, and V_θ using feed-forward networks. For task exploration, we use two separate $(128, 128, 128)$ networks with tanh activations to represent policy $\pi_\phi^{\text{explore}}$ and value function V_ϕ. These take 64-dimensional embeddings of s_t and $\Gamma^{(:k,:t)}$ as input, computed by a single-layer, tanh-activated network. We use similar architectures for π_θ and V_θ. However, z is first mapped to a 128-dimensional representation by a $(128, 128)$ feed-forward network with tanh activations.

To optimize both policies, each PPO update is run for $N_{\text{PPO}} = 4$ epochs, with minibatches of size 2. We use the Adam optimizer and set the coefficients in Eq. 10 to $c_1 = 0.5$ and $c_2 = 0.01$. The GAE $\hat{A}_t^{\text{GAE}}$ is computed using discounts $\gamma = 0.99$ and $\lambda = 0.9$. Finally, as mentioned in Sec. 3.1, we stabilize the optimization of π_θ by adding the PFO term suggested by Moalla et al. (2024) to the PPO objective (Eq. 10). We set the corresponding coefficient to $c_{\text{PFO}} = 0.1$. The PFO term is omitted when optimizing $\pi_\phi^{\text{explore}}$. We found that including it made training more difficult, whereas $\pi_\phi^{\text{explore}}$ was already sufficiently stable without it.

F.3 Meta-Training

LaSER agents are meta-trained in two phases. In the first phase, we alternate between updating the encoder g_ω and the exploration policy $\pi_\phi^{\text{explore}}$ for N_{explore} iterations. Before each encoder update, we collect a dataset $\mathcal{B}$ by following $\pi_\phi^{\text{explore}}$. Each element $\mathbf{B} \in \mathcal{B}$ is a tensor that contains Q meta-episodes and is collected from a task $\mathcal{M}_i \sim p(\mathcal{M})$. A dataset $\mathcal{B}$ contains data from 5 different tasks. We update g_ω on the masked dataset $\mathcal{B}^{\text{masked}}$ for N_{encoder} epochs, as shown in Alg. 3. To improve meta-training sample efficiency, we use all Q meta-episodes in each $\mathbf{B} \in \mathcal{B}$ to compute $\mathcal{L}_{\text{rec}}$ and $\mathcal{R}$. Additionally, in practice, each $\mathcal{B}$ is stored in a buffer and reused in future updates. This buffer holds data collected from up to $100,000$ tasks. To update $\pi_\phi^{\text{explore}}$, exploration data is collected from multiple tasks as described in Sec. 3.2. In the second phase, the task policy π_θ is updated for N_{task} iterations.

For the loss $\mathcal{L}_{\text{rec}}$ used in Eq. 9, we exclude the reconstructed actions from the loss computation. This ensures that the encoder focuses on learning patterns in states and rewards, rather than the exploration policy. Additionally, we scale the loss on states by a factor of 0.05.

Finally, note that the encoder g_ω is optimized purely through self-supervised methods. This constraint can be lifted. The RL loss used to update the task policy π_θ can also be used to fine-tune g_ω. This fine-tuning could adapt the general pre-trained encoder to align more closely with the task-solving objective. However, similar to Zintgraf et al. (2021b), we observe no empirical benefits from fine-tuning, so we omit it in our experiments.

G Baseline Algorithms Details

We provide details for the architectures, hyperparameters, and the meta-training process for the baseline algorithms used in Sec. 5.2 and 5.3. We tune these algorithms and use the best-performing hyperparameters. Where possible, and if performance is not negatively affected, the task policy's architecture and meta-training are similar to LaSER's. Otherwise, the task policy is tuned with the rest of the model.

G.1 MAML

For MAML (Finn et al., 2017), we use the publicly available code provided for meta-RL by Deleu (2018) at https://github.com/tristandeleu/pytorch-maml-rl. We train for 12000 meta-iterations, with batches of 16 tasks. For each task, we sample 5 episodes. MAML is meta-trained to maximize returns after one policy gradient update. However, during meta-testing, it performs $K = 4$ gradient updates. We use a discount factor of $\gamma = 0.99$. We also use $\lambda = 0.9$ to compute the generalized advantage estimator (GAE). To ensure optimal results, we did not use the first-order approximation proposed by Finn et al. (2017), but instead computed the second derivatives and backpropagated. The policy is a $(64, 64)$ feed-forward network with tanh activations. Due to the computational cost of MAML, we had to use a smaller policy network than in the other algorithms. All other hyperparameters follow the ones used by Finn et al. (2017) in their RL experiments.

G.2 PEARL

We use the publicly available code at https://github.com/katerakelly/oyster for our implementation of PEARL (Rakelly et al., 2019). Each agent is meta-trained for 350 iterations on a total of 3000 training tasks, with 16 tasks per batch. At each iteration, both the encoder and the task policy are optimized for 2000 gradient steps, with batches of 64 transitions for the encoder and 256 for the policy.

PEARL uses a variational approach for computing task representations z. When computing PEARL's loss, the KL divergence of the encoder is weighted by 0.1. Additionally, both the task policy and the encoder have a learning rate of 3×10^{-4} and use the Adam optimizer.

PEARL uses Soft-Actor Critic (SAC) (Haarnoja et al., 2018a;b), an off-policy RL algorithm. Note that, since SAC is designed for continuous action spaces, we instead use a SAC version for discrete action spaces (Christodoulou, 2019). We tune SAC's temperature hyperparameter automatically, using the approach introduced by Haarnoja et al. (2018b), since manual tuning can be difficult. We use a discount factor of $\gamma = 0.99$ and a target smoothing coefficient for SAC of 0.005. At each iteration, PEARL adds data collected from 5 randomly selected tasks to a replay buffer of size 1000000 timesteps. For each of these tasks, it collects 400 timesteps using a representation z sampled from a prior distribution over tasks. It additionally collects 600 more timesteps using a z sampled from its meta-trained posterior over tasks. However, this latter data is only used to update the task policy, not the encoder. Before training starts, the replay buffer is populated with 2000 timesteps per task, collected by following a uniformly random policy.

PEARL's encoder is a $(200, 200, 200)$ feed-forward network with ReLU activations that computes 5-dimensional latent task representations z. The task policy is a $(128, 128, 128)$ feed-forward network with tanh activations.

G.3 VariBAD

We use the code at https://github.com/lmzintgraf/varibad for VariBAD (Zintgraf et al., 2021b). We meta-train for 8125 iterations on 3000 meta-training tasks. At each iteration, we collect 200 timesteps per task from 16 different tasks. These timesteps are stored in a buffer of size 10000 episodes. Additionally, before training, a uniformly random policy adds 5000 timesteps to the buffer. This buffer is later used to update the encoder.

VariBAD uses a variational auto-encoder (VAE) (Kingma & Welling, 2014) to encode the data collected from tasks into latent representations z. This encoder is optimized using Adam with a learning rate of 0.001. At each iteration, the encoder is updated for 3 steps, using batches of 15 episodes, sampled from the buffer. The encoder is a recurrent neural network of size 128 that computes 5-dimensional task representations z. The decoder takes z as input, together with the current transition s, a, s', and reconstructs the reward r. We use a $(64, 32)$ feed-forward network to represent the encoder. The state, action, and reward inputs are pre-processed into representations of size 32, 16, and 16, respectively, by separate single-layer networks with ReLU activations. Both the encoder and decoder have such pre-processing layers.

Since VariBAD uses PPO to optimize its policy, we use the same network architectures and hyperparameters as LaSER's task policy (see Appendix F.2 and Table 5). However, we do not use the additional PFO objective. Finally, before passing s and z to the policy, we embed them into 64-dimensional representations using separate single-layer networks with tanh activations.

G.4 Hypernetwork Task Policy

We implement the hypernetwork-based task policy using the code provided by Beukman et al. (2024) at https://github.com/Michael-Beukman/DecisionAdapter. The policy is a $(128, 128)$ feed-forward network, similar to LaSER's task policy (see Appendix F.2). It is also meta-trained with the same hyperparameters (see Table 5). However, this policy only takes the state s as input. Moreover, between the policy's last hidden layer and output layer, we introduce an additional $(32, 32)$ network with tanh activations. The weights of this final network are generated by a hypernetwork, which takes the task representation z as input. Following Beukman et al. (2024), we also use a skip connection between the input and output of these hypernetwork-weighted layers.

The hypernetwork itself is a $(64, 64)$ feed-forward network with tanh activations. The output weights of the hypernetwork are computed in chunks of size 16. Before passing the input z, we pre-process it into a 4-dimensional representation using a single tanh-activated layer. Finally, the actor and the critic have separate hypernetworks.

H Hyperparameters

General Hyperparameters		
Hyperparameter	**Value**	**Notes**
N_{explore}	$10,000$	
H	50	Horizon.
K	4	Episodes per meta-episode.

Table 2: LaSER hyperparameters used throughout meta-training and meta-testing.

Encoder g_ω		
Hyperparameter	**Value**	**Notes**
Learning rate	1×10^{-4}	
N_{encoder}	50	
Q	20	Meta-episodes per task.
$c_{\text{rec}}, c_{\text{t-rec}}, c_{\text{contr}}, c_{\mathcal{R}}$	0.5; 1; 1; 0.125	Coefficients for loss $\mathcal{L}_{\text{LaSER}}$ in Eq. 9.
ξ	0.1	Constant in Eq. 7.
ν	250	Update delay for parameters $\hat{\omega}$.

Table 3: LaSER hyperparameters used for meta-training the encoder.

Exploration Policy $\pi_\phi^{\text{explore}}$		
Hyperparameter	**Value**	**Notes**
Learning rate	1×10^{-5}	
σ	0.0018	Constant in Eq. 4.
ϵ	0.1	PPO clipping during exploration; see Eq. 11.

Table 4: LaSER hyperparameters used for meta-training the exploration policy.

Task Policy π_θ		
Hyperparameter	**Value**	**Notes**
Learning rate	1×10^{-4}	
N_{task}	$20,000$	
β	1.5	Coefficient in Eq. 2.
ζ	-0.13	Threshold in Eq. 3.
ϵ	0.2	PPO clipping during task solving; see Eq. 11.

Table 5: LaSER hyperparameters used for meta-training the task policy.