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 also propose learning a meta-reward that encourages policies to learn to adapt. Empirical evaluations demonstrate that our algorithm 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 meta-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 settings. In few-shot adaptation, agents aim 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, meta-RL 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 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., 2022a), 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 still struggles 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 by using an implicit or explicit meta-learned exploration strategy. However, implicit exploration may be too sample inefficient for few-shot adaptation (e.g., agents that are not using their task distribution priors effectively 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). Moreover, the difficulties of few-shot adaptation extend beyond task exploration. Our empirical

results complement those of Beukman et al. (2024) in showing that optimizing a policy that leverages learned priors to solve new tasks can be non-trivial.

To get a clear high-level picture of in-context meta-RL, we consider separating it into three main parts, as shown in Fig. 1. The first phase is task exploration. After a task is sampled from the task distribution, a policy π^{explore} collects a "few" episodes of data from it. Optionally, this policy may be guided by an encoder f_u. The goal of π^{explore} is to collect data that contains useful information about the task. This information is then extracted by a second encoder g during the task learning stage, and encoded into a task context vector. Finally, in the task-solving phase, a policy π attempts to optimally solve the task at hand. This final stage is similar to standard RL, except that π is conditioned on the task context.

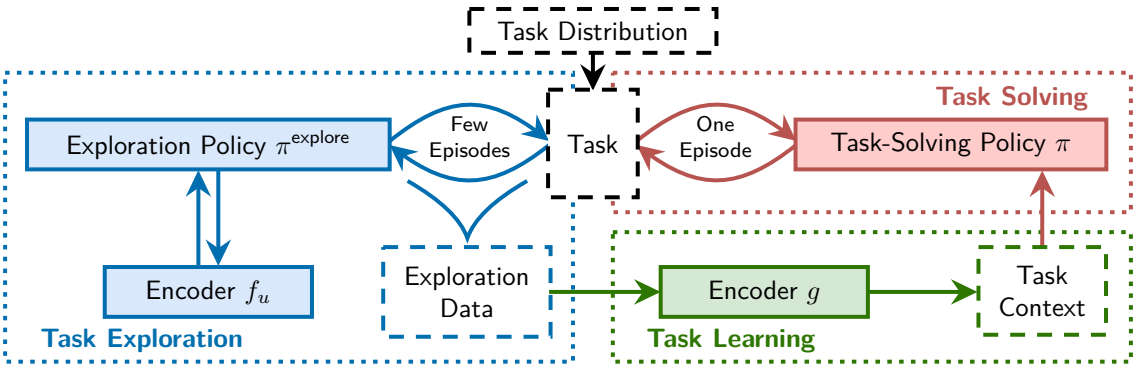

Figure 1: A high-level overview of in-context meta-RL.

There are several ways of learning to reinforcement learn. The approach previously described, and used in this work, is called in-context meta-RL. In-context policies π^{explore} and π use task context information provided by f_u and g to explore and solve tasks, respectively. Note that this three-phase paradigm is not always cleanly separated in practice, e.g., encoders f_u and g may share information or not be separate encoders at all, a single policy may be used for both task exploration and solving, etc. Nevertheless, this view helps form an intuitive understanding of meta-RL. Besides aiding intuition, we also use Fig. 1 throughout this paper to clarify which of the three components (i.e., task exploration, learning, or solving) we are discussing.

We introduce ***La****tent* ***S****pace* ***E****xploration via Task* ***R****econstruction* (***LaSER***), a novel meta-RL algorithm for few-shot adaptation.[1] The primary objective of LaSER is to learn task-specific exploration strategies. The core idea behind our method is to identify small datasets that capture rich task-specific information with minimal redundancy. We achieve this by introducing 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. We then use the context vectors encoded from this data to train a policy that solves tasks. However, we show that previous approaches to training such a policy fail, even when using optimal task contexts provided by an oracle. Moreover, this failure persists even when the task distribution is replaced by a predefined set of tasks. As a result, LaSER's secondary objective is to learn to use task contexts effectively. To achieve this, we propose an augmented RL objective that encourages the task policy to continue exploring until contexts are effectively exploited.

We summarize our main contributions as follows.

1. We formalize an important assumption about the data required for few-shot adaptation. This forms the basis of our task reconstruction method. We then show how an encoder built from these ideas can learn representations that facilitate exploration.

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

2. We leverage these representations to design an intrinsic reward for training a task-specific, sample-efficient exploration policy.

3. We propose using our aforementioned augmented objective to improve the adaptability of task-solving policies through directed exploration of the task context space. 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, while the latter offers richer task contexts.

We formalize the meta-RL problem and introduce the necessary background in Sec. 2. In Sec. 3, we address the objectives described above and introduce LaSER, our novel meta-RL algorithm. We position our work within the meta-RL literature in Sec. 4. In Sec. 5, we provide empirical results by evaluating 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 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 to adapt to new tasks. We consider a probability distribution $p(\mathcal{M})$ 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 $G(\tau) = \sum_{t=1}^{H} \gamma^t r_t$.

In-Context Meta-RL A common approach to solving meta-RL problems is in-context meta-RL, where a pretrained policy acts as an adaptable RL algorithm (Duan et al., 2016; Laskin et al., 2023; Moeini et al., 2025). Formally, given a context vector $\boldsymbol{c}_i$ that describes a task $\mathcal{M}_i$, the goal is to train a policy π to optimally solve $\mathcal{M}_i$ by taking in-context actions $a \sim \pi(\cdot \mid s, \boldsymbol{c}_i)$, conditioned on both the state s and the context $\boldsymbol{c}_i$. The meta-RL agent can therefore learn new policies from $\boldsymbol{c}_i$, even post-training. A standard approach is to concatenate s and $\boldsymbol{c}_i$, then optimize the in-context policy $\pi(a \mid s, \boldsymbol{c}_i)$ using standard RL algorithms. However, this simple method may not generalize well across tasks in complex task distributions (Beukman et al., 2024). To address this, we propose a novel meta-RL optimization algorithm that explicitly uses $\boldsymbol{c}_i$ to train in-context policies.

Meta-RL for Few-Shot Adaptation. In few-shot adaptation settings, in-context meta-RL attempts to optimize π to solve previously unseen tasks $\mathcal{M}_i \sim p(\mathcal{M})$ in a sample-efficient manner. However, because the meta-RL agent lacks knowledge of the task-specific dynamics T_i and R_i, it must first collect data from $\mathcal{M}_i$. The few-shot constraint limits this to K episodes per task. We call this a K-shot adaptation problem. Following Beck et al. (2023b), we refer to a sequence of K episodes collected in $\mathcal{M}_i$ as a meta-episode $\mathcal{D}_i^{(K)} = \{\tau_k\}_{k=1}^{K}$. We can then compute a latent context vector $\boldsymbol{c}_i = g(\mathcal{D}_i^{(K)})$ using an encoder g. Importantly, $\boldsymbol{c}_i$ should capture information about T_i and R_i, to allow generalization to new tasks. To collect such meta-episodes, we follow Liu et al. (2021) and define a separate exploration policy π^{explore}, decoupled from the task-solving policy π. For a given task, its goal is to collect an informative meta-episode, without necessarily achieving a high return. We define $\pi^{\text{explore}}(a \mid s, \boldsymbol{\Gamma}_i)$ to be an in-context policy, where $\boldsymbol{\Gamma}_i$ is a task representation encoded specifically for exploration. Although we can choose $\boldsymbol{\Gamma}_i$ to be the same as $\boldsymbol{c}_i$, it is less restrictive to use a new representation. Given this setup, we generalize Liu et al. (2021) and consider

agents that maximize the meta-RL objective

$$\mathcal{J}(\pi, \pi^{\text{explore}}, g) = \mathbb{E}_{\mathcal{M}_i \sim p(\mathcal{M}), \mathcal{D}_i^{(K)} \sim P_{\mathcal{M}_i}^{\pi^{\text{explore}}}} \left[V_i^{\pi} \left(g(\mathcal{D}_i^{(K)}) \right) \right], \tag{1}$$
$$\text{where } V_i^{\pi}(\boldsymbol{c}_i) = \mathbb{E}_{\tau \sim P_{\mathcal{M}_i}^{\pi(\boldsymbol{c}_i)}} \left[G(\tau) \right]$$

is the expected return in task $\mathcal{M}_i$ for the in-context policy π conditioned on $\boldsymbol{c}_i$.

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 $P_{\mathcal{M}_i}^{\pi^{\text{explore}}}$ 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 are represented as vectors or scalars. A timestep (s_t, a_t, r_t) is then 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. An episode $\tau \in \mathbb{R}^{H'}$ is a vector of H concatenated timesteps, where $H' = Hd$. 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. Episodes and meta-episodes can therefore be represented as tensors instead of sequences, when convenient.

3 Methods

In this section, we present our approach to in-context meta-RL and use the ideas we introduce to design the main components of our proposed algorithm, LaSER. We structure this section based on the three-stage meta-RL paradigm in Fig. 1. Sec. 3.1 corresponds to the task-solving phase, while Secs. 3.2 and 3.3 focus on the task exploration part. Sec. 3.4 continues task exploration but also discusses task learning, with Sec. 3.5 corresponding to the combination of these three stages into a complete meta-RL algorithm.

Specifically, in Sec. 3.1, we introduce a novel approach for in-context RL optimization. Given a context vector $\boldsymbol{c}_i$ for task $\mathcal{M}_i$, we propose augmenting the standard RL objective with a term that encourages meta-exploration whenever π can achieve better returns by improving its understanding of $\boldsymbol{c}_i$. Then, Sec. 3.2 discusses the training of an exploration policy π^{explore} for collecting the data required to compute $\boldsymbol{c}_i$. It assumes a given encoder f_u for training π^{explore}, with f_u modeled as a unidirectional transformer (Vaswani et al., 2017) of size d_{model}. We propose a novel method of training f_u for sample-efficient task-specific exploration in Sec. 3.3. We first introduce an important assumption on the data collected for few-shot adaptation, then show how a practical learning objective, i.e., task reconstruction, can be built from it. Next, we describe our proposed architecture for encoders f_u and g in Sec. 3.4. We model g as a transformer, as their ability to process each timestep in the context of an entire meta-episode has been shown to be advantageous in meta-RL settings (Melo, 2022; Shala et al., 2024). Finally, in Sec. 3.5, we introduce our meta-training and meta-testing algorithms. From this point forward, we simplify notation and drop the task subscript i whenever it is clear that we are referring to task-specific data or representations.

3.1 Task Solving

As discussed in Sec.2, Beukman et al. (2024) show that optimizing an in-context policy $\pi(a_t \mid s_t, \boldsymbol{c})$ to adapt to and solve tasks is non-trivial. We therefore introduce an approach for augmenting rewards with a term that encapsulates performance in the meta-RL setting. The agent can leverage this term to perform directed meta-exploration in the context of $\boldsymbol{c}$. With this new reward, we could overcome the aforementioned limitation by simply optimizing $\pi(a_t \mid s_t, \boldsymbol{c})$ using standard RL algorithms.

We start from the hypothesis that the task policy π is suboptimal because it fails to understand that $\boldsymbol{c}$ 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 meta-exploration objective. The policy $\pi(a_t \mid s_t, \boldsymbol{c})$ must understand the relationship between s_t and $\boldsymbol{c}$, at any timestep t where the optimal action depends on both s_t and $\boldsymbol{c}$. In other words, π needs to meta-explore more than in a standard RL setting.

Following this idea, we propose a method that encourages π to meta-explore more if the task context $\boldsymbol{c}$ is not properly used. Let $V(s_t, \boldsymbol{c})$ and $V(s_t)$ be the approximated value functions at state s_t, with and without considering $\boldsymbol{c}$, respectively. We make the assumption that, for an optimal policy, $V(s_t) \leq V(s_t, \boldsymbol{c})$ should hold at any timestep t. That is, using $\boldsymbol{c}$ should never decrease performance. If this does not hold for π in some state s_t, then the agent should meta-explore more from s_t. We adopt the standard approach of encouraging exploration by maximizing entropy (Williams & Peng, 1991), with the important distinction that this maximization is guided by the context $\boldsymbol{c}$. This idea is implemented as a meta-reward

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

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

is a non-negative dynamic weight on the entropy S of $\pi(\cdot \mid s_t, \boldsymbol{c})$, r_t is the environment reward, and β is a constant. Note that $w(s_t, \boldsymbol{c})$ is non-zero only when $V(s_t) - V(s_t, \boldsymbol{c}) \geq \zeta$, for a given threshold $\zeta \in (-\infty, 0]$.

Intuitively, $w(s_t, \boldsymbol{c}) > 0$ means that the agent is disregarding the information provided by $\boldsymbol{c}$ and should meta-explore more from state s_t. When $w(s_t, \boldsymbol{c}) = 0$, i.e., $V(s_t) \leq V(s_t, \boldsymbol{c}) + \zeta$, we recover the original RL objective, with $r_t^+ = r_t$. By simply replacing r_t with r_t^+, π can now be optimized in-context using standard RL algorithms. We use the proximal policy optimization (Schulman et al., 2017, PPO) algorithm in our work. Furthermore, we stabilize PPO (Wang et al., 2020; Sun et al., 2022; 2023; Moalla et al., 2024) using proximal feature optimization (Moalla et al., 2024, PFO). In Appendix A, we provide a short overview of PPO and explain our method of applying it to meta-RL.

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

3.2 Task Exploration Policy

To create a context $\boldsymbol{c}$ for identifying a task, the meta-RL agent must first learn to explore 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 that π has enough prior knowledge about the task distribution $p(\mathcal{M})$.

The encoder f_u computes a matrix $\boldsymbol{\Gamma} = f_u(\mathcal{D}^{(K)}) \in \mathbb{R}^{(Hd_{\text{model}}) \times K}$. Here, $\boldsymbol{\Gamma}$ is a latent representation of the K episodes in $\mathcal{D}^{(K)}$, with $\boldsymbol{\Gamma}_{:,k}$ denoting the representation of the k-th episode $\mathcal{D}_{:,k}^{(K)}$. We optimize f_u such that the similarity between $\boldsymbol{\Gamma}_{:,k}$ and $\boldsymbol{\Gamma}_{:,k'}$ 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 using the pairwise cosine similarity matrix $S_C(\boldsymbol{\Gamma}^\mathsf{T}, \boldsymbol{\Gamma}) \in \mathbb{R}^{K \times K}$. Lower scores correspond to lower similarity between the K episodes, which in turn corresponds to $\mathcal{D}^{(K)}$ being more informative. In Sec. 3.3, we show how to meta-train a function f_u with this property.

We use this idea to optimize π^{explore} for task exploration. We begin by computing the average column-wise similarity in $\mathcal{D}^{(K)}$ as a vector $\boldsymbol{d} = \frac{1}{K} S_C(\boldsymbol{\Gamma}^\mathsf{T}, \boldsymbol{\Gamma})^\mathsf{T} \mathbf{1} \in \mathbb{R}^K$. Consequently, $\boldsymbol{d}_k$ represents the average similarity between the k-th episode and all other episodes. Therefore, a meta-episode $\mathcal{D}^{(K)}$ 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

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

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

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 meta-trained on full meta-episodes. Therefore, we apply a causal mask when computing $\boldsymbol{d}$ to hide similarities between the representation of an episode $\mathcal{D}_{:,k}^{(K)}$ and those of any future episodes $\mathcal{D}_{:,k'}^{(K)}$, for $k < k'$.

For π^{explore} to collect a useful episode, it must have knowledge of the episodes it has already collected. To create an exploration-focused task context, 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 the in-context policy π^{explore} on the history $\boldsymbol{\Gamma}^{(:k,:t)} = f_u(\mathcal{D}^{(:k,:t)})$ and explore episode k by taking action $a_{t+1}^k \sim \pi^{\text{explore}}(\cdot \mid s_{t+1}^k, \boldsymbol{\Gamma}^{(:k,:t)})$. The overall task exploration process is visualized in Fig. 2.

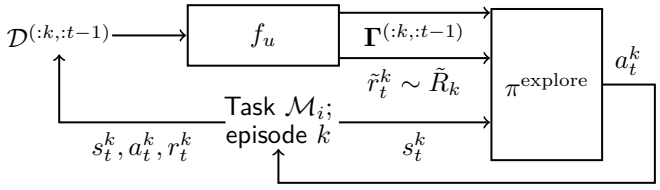

Figure 2: Task exploration. Policy π^{explore} explores a task $\mathcal{M}_i$ by collecting K episodes. At each timestep t of each episode k, π^{explore} is conditioned on the current state s_t^k and the latent representation $\boldsymbol{\Gamma}^{(:k,:t-1)}$ of previous timesteps and episodes. Its objective is to take actions that maximize the intrinsic exploration reward function $\tilde{R}_k$. Both $\boldsymbol{\Gamma}^{(:k,:t-1)}$ and $\tilde{R}_k$ are computed by the encoder f_u from the data collected.

3.3 Learning to Explore 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 $\boldsymbol{c} = g(\mathbf{B}_j)$ for task-solving and $\boldsymbol{\Gamma} = f_u(\mathbf{B}_j)$ 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. Before describing this assumption, we first formalize the idea of K-shot adaptation.

Definition 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 is K-shot adaptable in a task $\mathcal{M}_i$ when using the QK episodes in $\boldsymbol{B}_{[2]}$, Definition 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 1 (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 1 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 1. 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 context vector $\boldsymbol{c}$ 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 $\{1, 2, \ldots, Q\}$, 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

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

where ξ is a constant and the operator $\overline{(\cdot)}$ gives the element-wise mean of any tensor. This target measures the linear task reconstruction in Eq. 5, such that $\delta_j \approx 0$ implies $\mathbf{B}_j$ 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 $\mathbf{\Gamma}$, we optimize the similarity between any two episode representations $\mathbf{\Gamma}_{:,k}$ and $\mathbf{\Gamma}_{:,k'}$ to be δ_j, for $k, k' \in [K]$. As in Sec. 3.2, we use cosine similarity. We can now define the loss

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

and $\boldsymbol{A}(\delta_j) \in \mathbb{R}^{K \times K}$ is the matrix in which off-diagonal elements are δ_j and diagonal elements are 1.

Eq. 8 is a form of contrastive learning for RL (Eysenbach et al., 2022; Erraqabi et al., 2022), where $\mathbf{\Gamma}_{:,k}$ and $\mathbf{\Gamma}_{:,k'}$ are pushed apart or pulled together according to a soft similarity signal δ_j. This signal quantifies the similarity between the corresponding inputs $\mathbf{B}_{j,:,k}$ and $\mathbf{B}_{j,:,k'}$. Since this contrastive objective operates over cosine similarity, pushing dissimilar representations corresponds to maximizing the angle between them, while pulling corresponds to aligning similar representations by minimizing the angle. Intuitively, given the definition of similarity in Eq. 7, the angles between the K columns in $\mathbf{\Gamma}$ are optimized to encode how well $\mathbf{B}_j$ reconstructs the task data $\mathbf{B}$. Therefore, collecting a meta-episode where all vectors in $\{\mathbf{\Gamma}_{:,k}\}_{k=1}^{K}$ are approximately orthogonal to each other is now equivalent to searching for a $\mathbf{B}_j$ that closely approximates $\mathbf{B}$. The important advantage is that this method of finding $\mathbf{B}_j$ can be used for K-shot adaptation. In contrast, the alternative approach of collecting Q meta-episodes would be much more sample-inefficient. The encoder f_u can now be used to train an exploration policy π^{explore} to find a meta-episode $\mathbf{B}_j$, as described in Sec. 3.2.

3.4 Architecture and Optimization

We now introduce the architecture of our encoder g. Because of their close relation, 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 task exploration. 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). To obtain more informative contexts once all task data has been collected, we define an additional bidirectional transformer f_b of size d_{model}, with parameters ω_b. Finally, we set g to be composed of f_u and f_b.[3]

[3]Due to overlapping terminology, LaSER can be seen as performing in-context learning in two senses. From the meta-RL perspective, LaSER uses in-context policies conditioned on contextual information. From a different perspective, these context vectors are computed using transformers, which, due to their self-attention mechanism, are considered to be in-context learners.

Given $\mathbf{B}$, f_b computes two new latent representations, $\boldsymbol{z}$ and $\mathbf{Z}$, as $(\boldsymbol{z}, \mathbf{Z}) = f_b(\mathbf{B}; \omega_b)$. Importantly, f_b is bidirectional, so it encodes data "offline", i.e., after task exploration is over. Although $\mathbf{B}$ refers to a single-task collection of meta-episodes, in practice, we meta-train on a batch of such tensors, each collected from a different task. This batching accelerates learning. More importantly, it allows us to capture the shared task structure across $p(\mathcal{M})$ by computing the vector $\boldsymbol{z} \in \mathbb{R}^{d_{\text{model}}}$ from the full batch of meta-episodes. In contrast, the tensor $\mathbf{Z} \in \mathbb{R}^{Q \times HK \times d_{\text{model}}}$ is a task-specific representation of $\mathcal{M}_i$, encoded solely from the dataset $\mathbf{B}$ collected in $\mathcal{M}_i$. We use $\mathbf{Z}_j$ to denote the matrix $\mathbf{Z}_{j,:,:} \in \mathbb{R}^{(HK) \times d_{\text{model}}}$ encoding the meta-episode $\mathbf{B}_j$, for any $j \in [Q]$. While $\mathbf{Z}$ and $\boldsymbol{\Gamma}$ have similar roles, $\boldsymbol{\Gamma}$ is optimized to learn representations that are only useful for task exploration, so it may fail to capture structure that is important for solving tasks, but irrelevant for exploration.

For a given meta-episode $\mathbf{B}_j$, we compute the task context vector $\boldsymbol{c} \in \mathbb{R}^{(HKd_{\text{model}})}$ as a function of $\boldsymbol{z}$, $\mathbf{Z}_j$, and $\boldsymbol{\Gamma}$.[4] Specifically, for a function h_{ω_h} with parameters ω_h, we encode $\mathbf{B}_j$ into

$$\boldsymbol{c} = g_\omega(\mathbf{B}_j) = h_{\omega_h}(\boldsymbol{z}, \mathbf{Z}_j, \boldsymbol{\Gamma}), \tag{9}$$

where g_ω has the concatenated parameters $\omega = \omega_u \oplus \omega_b \oplus \omega_h$. In practice, h consists of two attention layers. Our encoder architecture is shown in Fig. 3.

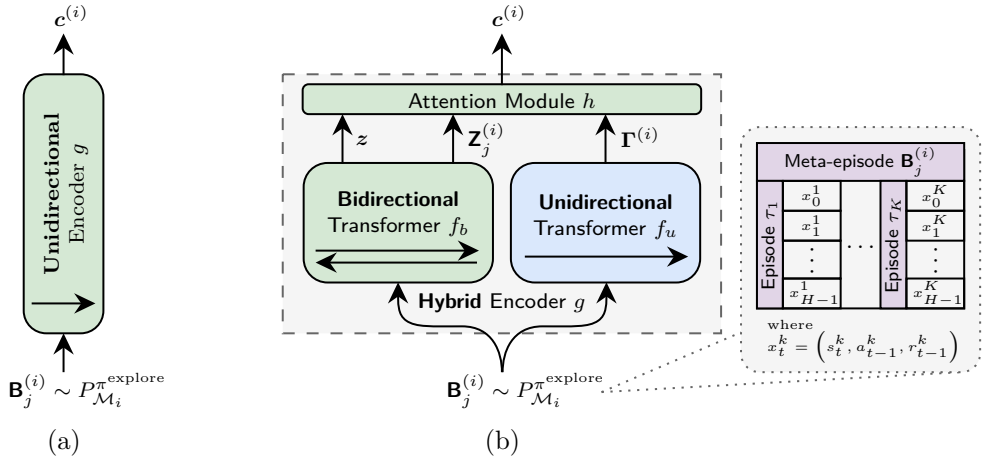

Figure 3: Two types of meta-RL encoders. (a) Unidirectional encoder, common in meta-RL. It processes task exploration data online, step by step, as it is being collected. (b) LaSER's encoder g. We enhance standard architectures by adding a bidirectional encoder. Encoder g can be used online for task exploration or offline to compute context $c^{(i)}$. We use $\cdot^{(i)}$ to differentiate between meta-episodes and representations belonging to different tasks. Note that we define timesteps x_t^k to contain the current state s_t^k, but the previous action a_{t-1}^k and reward r_{t-1}^k, with $x_1^k = (s_1^k, \_, \_)$. This simplifies the training of unidirectional encoders.

To train our encoder g, 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 coefficient tensor $\mathbf{C} = h_{\text{t-rec}}(g_\omega(\mathbf{B}), g_\omega(\mathbf{B}_j); \omega_{\text{t-rec}})$ required in Eq. 6. That is, $h_{\text{t-rec}}$ combines the representations of Q meta-episodes, $g_\omega(\mathbf{B})$, with the representation of the j-th meta-episode, $g_\omega(\mathbf{B}_j)$. Note that, to increase training stability, the tensor $\mathbf{C}$ used to compute δ_j in Eq. 7 is encoded by target parameters $\hat{\omega}_{\text{t-rec}}$, which are only updated to $\omega_{\text{t-rec}}$ every ν iterations (Mnih et al., 2015). The latter head is used during the optimization of g_ω to reconstruct data from corrupted inputs. This is a common approach for pre-training bidirectional transformers. Since both $h_{\text{t-rec}}$ and h_{rec} are only used during meta-training, they are replaced by policies π^{explore} and π during meta-testing. Fig. 4 gives an overview of the heads.

[4]The reader might note that the (HKd_{model})-dimensional context vector $\boldsymbol{c}$ does not necessarily reduce the size of the input $\mathbf{B}_j$. A high-dimensional $\boldsymbol{c}$ is beneficial since preserving the structure of $\mathbf{B}_j$ (i.e., the HK timesteps) makes it simpler to train transformers f_u and f_b. Despite this, $\boldsymbol{c}$ is still a useful representation of $\mathbf{B}_j$. Each of the HK output timesteps has been computed from its corresponding input timestep and the rest of the input sequence. This enables dimensionality reduction later on. Specifically, we use the task policy π to reduce $\boldsymbol{c}$ to a d_{model}-dimensional vector.

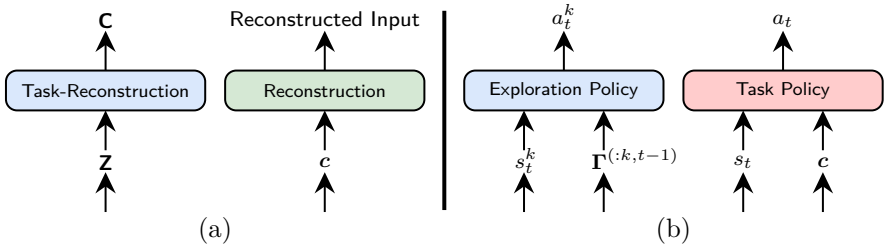

Figure 4: LaSER's pre-training and policy heads. (a) Task-reconstruction head $h_{\text{t-rec}}$ and reconstruction head h_{rec}, used for pre-training encoder g for task exploration and task learning, respectively. (b) Policies π^{explore} and π, used for task exploration and task solving, respectively.

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

$$\mathcal{L}_{\text{LaSER}}(\omega, \omega_{\text{rec}}, \omega_{\text{t-rec}}) = c_{\text{rec}}\mathcal{L}_{\text{rec}}(\omega, \omega_{\text{rec}}) + c_{\text{t-rec}}\mathcal{L}_{\text{t-rec}}(\omega, \omega_{\text{t-rec}}) + c_{\text{contr}}\mathcal{L}_{\text{contr}}(\omega) + c_{\mathcal{R}}\mathcal{R}(\omega), \tag{10}$$

where $c_{\text{rec}}, c_{\text{t-rec}}, c_{\text{contr}}, c_{\mathcal{R}}$ are weighting 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 $\boldsymbol{z}$ and $\mathbf{Z}_{j,t,:}$, for each $j \in [Q]$ and timestep $t \in [HK]$, by minimizing $(\boldsymbol{z}^\top \mathbf{Z}_{j,t,:})^2$. Intuitively, $\mathbf{Z}$ is encouraged to avoid capturing information already contained in $\boldsymbol{z}$, and instead focus on task-specific structure. Furthermore, $\mathcal{R}$ normalizes the vector representations $\boldsymbol{z}$, $\mathbf{Z}_{j,t,:}$, and $\mathbf{\Gamma}_{:,k}$, for all $j \in [Q], t \in [HK], k \in [K]$.

3.5 Meta-Training and Meta-Testing

The LaSER meta-training algorithm is shown in Alg. 1. It contains two training phases. We first train the exploration policy $\pi_\phi^{\text{explore}}$, parameterized by ϕ, and the encoder g_ω, parameterized by ω, for N_{explore} iterations. 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 for N_{task} iterations.

Algorithm 1 LaSER Meta-Training

Input $p(\mathcal{M})$, task distribution
Output π_θ, task policy; $\pi_\phi^{\text{explore}}$, exploration policy; g_ω, encoder; $\hat{\boldsymbol{z}}$, shared component

$\theta, \phi, \omega, \omega_{\text{rec}}, \omega_{\text{t-rec}} \leftarrow$ initialize randomly
for $n \in [N_{\text{explore}}]$ **do**
  $\omega, \omega_{\text{rec}}, \omega_{\text{t-rec}} \leftarrow$ train_encoder$(p(\mathcal{M}), \pi_\phi^{\text{explore}}, g_\omega, \omega_{\text{rec}}, \omega_{\text{t-rec}})$ ▷ Alg. 3
  $\phi \leftarrow$ train_exploration_policy$(p(\mathcal{M}), \pi_\phi^{\text{explore}}, g_\omega)$ ▷ Alg. 4
end for
$\mathcal{B} \leftarrow \left\{ \mathbf{B} \sim P_{\mathcal{M}_i}^{\pi_\phi^{\text{explore}}} \,\middle|\, \mathcal{M}_i \sim p(\mathcal{M}) \right\}$ ▷ Collect a batch of data from multiple tasks
$\hat{\boldsymbol{z}}, \_ \leftarrow f_b(\mathcal{B}; \omega_b)$
for $n \in [N_{\text{task}}]$ **do**
  $\theta \leftarrow$ train_task_policy$(p(\mathcal{M}), \pi_\theta, \pi_\phi^{\text{explore}}, g_\omega, \hat{\boldsymbol{z}})$ ▷ Alg. 5
end for
return $\pi_\theta, \pi_\phi^{\text{explore}}, g_\omega, \hat{\boldsymbol{z}}$

Recall that $\boldsymbol{z}$ is computed from multiple tasks, which is only feasible during the pre-training phase. Therefore, we compute a fixed $\hat{\boldsymbol{z}}$ from the pre-training data and use it to train the task policy and perform meta-testing.

During meta-testing, for any $\mathcal{M}_i \sim p(\mathcal{M})$, the meta-trained agent first collects K episodes and computes the latent context $\boldsymbol{c}$. Then, it uses $\boldsymbol{c}$ to find the optimal policy for $\mathcal{M}_i$. This is shown in Alg. 2.

Algorithm 2 LaSER Meta-Testing

Input $p(\mathcal{M})$, task distribution;
π_θ, task policy; $\pi_\phi^{\text{explore}}$, exploration policy; g_ω, encoder;
$\hat{\boldsymbol{z}}$, shared component

1: **for** $\mathcal{M}_i \sim p(\mathcal{M})$ **do**
2: $\quad \mathcal{D}^{(K)} \sim P_{\mathcal{M}_i}^{\pi_\phi^{\text{explore}}}$ ▷ Sample exploration meta-episode
3: $\quad \boldsymbol{c} \leftarrow g_\omega(\mathcal{D}^{(K)})$ ▷ Compute $\boldsymbol{c}$, using $\hat{\boldsymbol{z}}$ as the shared component
4: $\quad \tau \sim P_{\mathcal{M}_i}^{\pi_\theta(a|s,\boldsymbol{c})}$ ▷ Sample exploitation episode
5: $\quad$ Measure return in τ
6: **end for**

4 Related Work

Meta-RL. The earlier successes of modern meta-RL start with MAML (Finn et al., 2017) and RL^2 (Duan et al., 2016; Wang et al., 2016a). MAML, together with other MAML-inspired algorithms (Sung et al., 2017; Li et al., 2017; Gupta et al., 2018; Zintgraf et al., 2019), are gradient-based methods for meta-training policies that can adapt to new tasks by taking a small number of task-specific gradient steps. Gradient-based approaches usually implement an explicit dual-loop algorithm that follows the standard meta-learning paradigm of adapting to tasks in an inner loop, while meta-learning adaptation strategies in an outer loop. On the other side of the spectrum, RL^2 is an in-context meta-RL approach, meta-training a policy to behave like an RL algorithm that learns from collected task contexts (Laskin et al., 2023; Moeini et al., 2025). This learning is usually represented by a forward pass through the meta-trained policy, with no task-specific weight updates. Recent meta-RL techniques, including ours, follow this paradigm of first identifying task dynamic, and then adapting a task-agnostic policy to them. Generally, this in-context learning tends to be more sample-efficient than gradient-based methods, which is ideal in few-shot adaptation (Beck et al., 2023b).

In-Context Policies. To learn task-specific policies, RL^2 uses a recurrent architecture that implicitly conditions the policy on task history. Later works make this conditioning more explicit (Rakelly et al., 2019; Zhou et al., 2019; Zintgraf et al., 2019). Specifically, they show that in-context policies can be trained using standard RL optimization by simply augmenting the input state with task contexts. Similar ideas have been explored in the closely related area of unsupervised representation learning for RL (Igl et al., 2018; Papoudakis et al., 2021; Botteghi et al., 2025). This approach has since become standard in meta-RL, with subsequent research focusing more on learning informative task contexts than on novel architectures or optimization algorithms for in-context policies. Recently, Beukman et al. (2024) observed that this simple method may struggle when there is high variation across the optimal task-specific policies, which can arise in complex task distributions. As an alternative, Beck et al. (2023a) and Beukman et al. (2024) propose that meta-training hypernetworks (Ha et al., 2017) may lead to better generalization. Therefore, they introduce hypernetworks that take the task context as input and generate the weights of a context-dependent policy. While this architectural change may better leverage task contexts, their approach is still limited by the use of standard RL optimization, which constrains task space exploration. Additionally, this method inherits issues related to hypernetworks, such as slow and unstable training (Ortiz et al., 2023; Chauhan et al., 2024; Beukman et al., 2024). In contrast, our proposed meta-reward is not an adaptation of standard learning methods to meta-RL, but is explicitly designed for the meta-RL setting. Moreover, it is architecture-agnostic, introduces little overhead, and allows in-context policies to be optimized through standard RL. This may simplify solving meta-RL problems, as practitioners can rely on stable and well-understood RL algorithms.

Exploration in Meta-RL. A considerable body of literature also focuses on exploration in meta-RL. As opposed to standard RL, exploration strategies for few-shot meta-RL 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 make exploration more efficient by structuring the exploration space through 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). 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). LaSER also collects data using a decoupled exploration policy and then meta-learns a structured exploration space. However, it uses a novel objective that encourages the collection of a single meta-episode, which serves as a low-rank linear representation of a larger dataset drawn from the same task. An important distinction to numerous previous works is that task exploration depends only on the structure of the data, while being agnostic to the RL objective of the task policy. This may be advantageous for out-of-distribution adaptation, where meta-test tasks may have goals that differ from those seen in meta-train tasks. Therefore, data collected for maximizing meta-training return might not always be relevant during meta-testing.

Meta-Learning Contexts. Once task data has been collected, a straightforward way to obtain informative contexts is through recurrent neural networks (Duan et al., 2016; Wang et al., 2016a). More sophisticated methods 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 in-context and 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 aligns well with the needs of in-context meta-RL. Earlier research focused solely on meta-learning through attention (Mishra et al., 2018), while more recent work used transformer architectures (Melo, 2022; Xu et al., 2022b; Lee et al., 2023; Shala et al., 2024; Grigsby et al., 2024). A limitation of previous works that attempt to meta-learn task dynamics is that only unidirectional encoders are used. This constraint arises naturally since task exploration and task learning are coupled. Specifically, task data must be encoded online while it is being collected, in order to guide the exploration policy at the next timestep. An obvious limitation is that only interactions between the current and past timesteps are considered. LaSER improves this design by using an additional bidirectional encoder that also considers future timesteps, which could possess useful information and lead to richer representations (Devlin et al., 2019; Banino et al., 2022). While the aforementioned constraint cannot be avoided during task exploration, it need not restrict the computation of the final task context once all exploration data has been collected.

5 Experiments

In this section, we present empirical results for LaSER. We first introduce the environment and algorithms we use in Sec. 5.1. In Sec. 5.2, we compare LaSER with other types of meta-RL algorithms on multiple benchmarks. Next, we perform two ablation studies by analyzing individual stages of the meta-RL pipeline (Fig. 1). In Sec. 5.3, we evaluate our novel approach to meta-training in-context task policies. This corresponds to LaSER's task-solving phase, which we perform using ground-truth contexts instead of task exploration and learning. Finally, we assess LaSER's task exploration and task learning stages in Sec. 5.4 by visualizing the latent task contexts computed during meta-testing.

5.1 Experimental Setup

5.1.1 Environments

We evaluate LaSER on the meta-RL benchmarks MEWA (Stoican et al., 2023), Meta-World (Yu et al., 2020; McLean et al., 2025), and MuJoCo HopperMass (Rothfuss et al., 2019; Zhou et al., 2019; Nakhaeinezhadfard et al., 2025). We briefly discuss these environments here, focusing especially on MEWA. We then provide more details in Appendix E, including the publicly available code used to implement each benchmark.

MEWA provides a distribution of tasks that share the same central idea: certain states, called critical states, can lead to mistakes. These mistakes affect the final return negatively. The probability of a mistake happening depends on both the type of critical state and the task's dynamics. For each task, meta-RL agents must find the optimal policy for avoiding high-risk mistakes while minimizing delays. Importantly, Stoican et al. (2023) ensure MEWA's task distribution has no globally optimal non-adaptive policy (i.e., no policy can zero-shot solve all tasks). Therefore, the optimal agent must collect new data and adapt. This property makes MEWA ideal for our case, allowing us to evaluate LaSER's task exploration ability.

MEWA evaluates agents on their ability to take optimal actions in different types of "critical" states. These critical states provide agents with two options. Consider a critical state s_x of type x. The agent's first option is to take a "risky" action. This may lead to a mistake of type x happening, which in turn leads to a large delay in task completion. The probability of a type x mistake happening depends on the transition function of the task. The second option is a "safe" action. This provides a guaranteed small delay and leads to a state s_{x-1} of type x. An important feature shared by all MEWA tasks is that for any $y < x$, mistakes of type y are less likely to happen than mistakes of type x. Therefore, depending on the task, taking several safe actions before a risky action could be optimal.

We meta-train all meta-RL agents on the narrow task distribution analyzed by Stoican et al. (2023). This corresponds to a distribution $p(\mathcal{M})$ in which any task $\mathcal{M}_i \sim p(\mathcal{M})$ has four different types of critical states and can be described by a vector $\boldsymbol{p}^{(i)} \in [0,1]^4$. For each task $\mathcal{M}_i$, the probability of making a mistake of type $x \in [4]$ is $\boldsymbol{p}_x^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}_x, 0.12)$, where $\mathcal{N}$ is a normal distribution and $\boldsymbol{\mu}^\mathsf{T} = [0.38, 0.28, 0.19, 0.09]$. Note that we clip all $\boldsymbol{p}_x^{(i)}$ to $[0,1]$. See Appendix E.1 for a more formal definition of MEWA's tasks.

We meta-test on the fixed set of 12 tasks proposed by Stoican et al. (2023). Their corresponding $\boldsymbol{p}^{(i)}$ vectors are listed in Appendix E.1, Table 3. We assign a score to each task, given by the average probability of a mistake of any type happening, i.e., $1/4 \sum_{x=1}^{4} \boldsymbol{p}_x^{(i)}$. We use this as a rough estimate of the similarity between tasks, such that tasks with a similar score require similar (but not identical) optimal policies. Additionally, note that half of the meta-testing tasks are out-of-distribution (OOD) tasks. OOD tasks lie outside the meta-training distribution $p(\mathcal{M})$, so they are more challenging to explore and solve during meta-testing.

Note that no globally optimal non-adaptive policy exists for these 12 meta-testing tasks. Additionally, 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.

MEWA offers 3 baselines to compare our agents to during meta-testing. 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 value function V^*, if $V^*(s, \boldsymbol{c}) \leq V^*(s)$ for all $\boldsymbol{c}$, 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 gives the expected return of the optimal meta-RL agent, i.e., the agent that optimally adapts to and solves each task.

To assess the generalizability of our algorithm, we also conduct experiments on Meta-World V2 ML10 and MuJoCo HopperMass. Meta-World is a suite of 10 meta-training and 5 meta-testing robotic manipulation tasks where the goal is hidden and must be meta-learned. To avoid reproducibility issues, we follow McLean et al. (2025) and use the V2 reward function. HopperMass is the meta-RL extension of the popular MuJoCo Hopper environment (Todorov et al., 2012). The agent's goal is to rapidly move forward, taking into account

task variations given by multiplying the body parts' mass, the joints' damping, and the ground friction with weights uniformly sampled from $[1.5^{-1}, 1.5^{1}]$. To assess out-of-distribution (OOD) generalization, we follow Nakhaeinezhadfard et al. (2025) and extend HopperMass to OOD meta-testing tasks with weights in $[1.5^{-1.5}, 1.5^{-1}) \cup (1.5^{1}, 1.5^{1.5}]$.

In some environments, such as HopperMass, low exploitation returns during task exploration may lead to the LaSER agent only "surviving" for a small number of timesteps, hindering effective data collection. To avoid exploratory episodes terminating prematurely, we let LaSER's reward function for episode k be

$$\tilde{R}'_k(s^k_t, a^k_t) = \tilde{R}_k(s^k_t, a^k_t) + \alpha_R R(s^k_t, a^k_t), \tag{11}$$

where $\tilde{R}$ is the exploration reward function in Eq. 4, R is the environment reward function, and α_R is a constant weight. For MEWA and Meta-World, we set $\alpha_R = 0$. In HopperMass, we tune α_R such that R does not dominate $\tilde{R}$.

5.1.2 Algorithms and Meta-Training Setup

We compare our method, LaSER, with four other meta-RL algorithms, MAML (Finn et al., 2017), PEARL (Rakelly et al., 2019), VariBAD (Zintgraf et al., 2021b), and DREAM (Liu et al., 2021). For each benchmark, we meta-train the algorithms on tasks sampled from the meta-training distribution. For each task $\mathcal{M}_i$, the agents are allowed to collect K episodes during the task exploration phase. For both MEWA and HopperMass, we use $3,000$ tasks with $K = 4$, while for Meta-World we use the 10 tasks provided and set $K = 2$. The agents' performance on $\mathcal{M}_i$ is then given by a single exploitation episode collected by the task policy. Besides this final return, we also analyze an agent's ability to improve its performance as more data is collected. MEWA tasks have a horizon of $H = 50$, Meta-World has $H = 200$, and HopperMass has $H = 400$. All of the results we present are averaged over 10 seeds.

We meta-train LaSER in two decoupled phases. Following Alg. 1, the exploration policy $\pi^{\text{explore}}_\phi$ and encoder g_ω are first trained for N_{explore} iterations. The task policy π_θ is then trained separately for N_{task} iterations. Note, however, that for the first $N^{\text{initial}}_{\text{explore}}$ out of N_{explore} iterations, we only train f_u (and thus g_ω), without updating $\pi^{\text{explore}}_\phi$ since its training is conditioned on representations learned by f_u. For these initial updates, we use data collected by a uniformly random policy.

LaSER's hyperparameters are listed in Appendix H. We selected these based on performance on the meta-training task distribution of each environment. For implementation details and hyperparameters for MAML, PEARL, VariBAD, and DREAM, see Appendix F.

Note that we perform our primary in-depth analysis on MEWA, as its design ensures that task exploration and adaptability are strictly necessary. In contrast, while Meta-World and HopperMass are important benchmarks, they have not been explicitly designed to test task exploration (Liu et al., 2021). Therefore, their primary role in our work is to demonstrate the general applicability of our algorithm to meta-RL problems. For these two benchmarks, we use VariBAD as the baseline. This choice is justified as VariBAD outperforms MAML and PEARL on Meta-World (Zintgraf et al., 2021b), and DREAM was not designed for this type of benchmark (Liu et al., 2021).

5.1.3 Meta-Training Complexity

We analyze the computational complexity of meta-training each algorithm on MEWA. Specifically, we measure the total wall clock time required to collect environmental transitions and optimize all components of an algorithm. We additionally report the total time divided by the number of timesteps collected, showcasing the average cost of interacting with the environment once and updating based on that interaction. Note that all seeds of an algorithm are meta-trained for the same number of timesteps. This duration is set by the timestep at which the last seed's performance (i.e., return or loss) stops improving.

We provide detailed measurements, averaged over 10 seeds, in Fig. 15. Regarding total time, LaSER is the most expensive, being 1.34 times slower than the next slowest algorithm, DREAM. This is followed by PEARL, and finally by MAML and VariBAD. Most of LaSER's time complexity comes from its transformer

encoders, which are powerful but require a high amount of data. However, LaSER appears more efficient when collecting a single timestep and updating. Specifically, LaSER is 11.93 and 5.86 times faster than DREAM and PEARL, respectively. Furthermore, it is only 1.71 and 2.64 times slower than MAML and VariBAD, respectively.

To understand the architectural complexity of LaSER, Table 4 compares the number of trainable parameters to those of the baseline algorithms. While LaSER has notably more parameters, approximately 90% of these belong to the two pre-training heads, h_{rec} and $h_{\text{t-rec}}$. Since these are not used beyond the pre-training phase, LaSER agents only require approximately 8 million parameters during meta-testing and task policy meta-training. This is roughly eight times larger than DREAM, the next-largest architecture.

LaSER's high number of parameters is a consequence of the transformer encoder g, whose additional complexity enables more efficient in-context computations. Moreover, since the context vector $\boldsymbol{c}$ outputted by g is also higher-dimensional than the ones used by PEARL, VariBAD, and DREAM, LaSER's task policy requires a larger network for extracting features from $\boldsymbol{c}$. Besides this feature extractor, LaSER's policy networks are comparable in size to those of PEARL and VariBAD.

5.2 Results and Analysis

5.2.1 Meta-Training Convergence on MEWA

To ensure a fair evaluation, we first show that all algorithms converge during meta-training. For this, we evaluate agents on MEWA's meta-training task distribution. As shown in Fig. 5, LaSER's exploration policy, encoder, and task policy converge across 10 seeds. The encoder loss is computed by Eq. 10, with Fig. 10 showing results for each of its terms. The exploration return is computed using intrinsic rewards (see Eq. 4) collected by the exploration policy. Note that we first train the encoder and exploration policy in parallel for 220 and 170 million timesteps, respectively, then the task policy for 7.5 million timesteps.

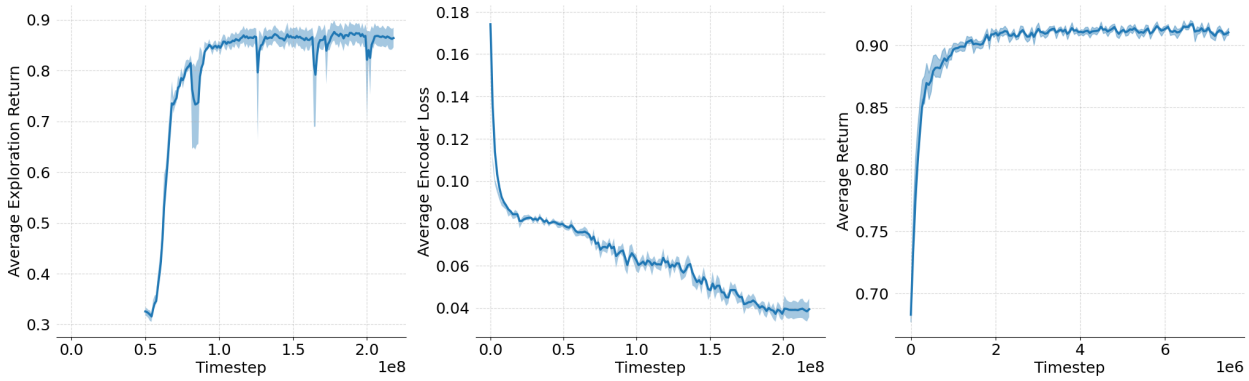

Figure 5: Performance on tasks from MEWA's meta-training task distribution. For each metric, the shaded areas report the standard error of the average return across 10 random seeds. We use exponential moving average (EMA) smoothing $y_j = \alpha_{\text{EMA}} x_j + (1 - \alpha_{\text{EMA}}) y_{j-1}$ for each point x_j, with $\alpha_{\text{EMA}} = 0.6$. We first meta-train the exploration policy and the encoders for $2.2e8$ timesteps, then the task policy for $7.5e6$. Additionally, we only start exploring tasks after 50 million timesteps. (left) Undiscounted exploration return of policy $\pi_\phi^{\text{explore}}$ for exploration reward function $\tilde{R}$. (middle) Encoder loss $\mathcal{L}_{\text{LaSER}}$. (right) Undiscounted return of policy π_θ.

Appendix G provides convergence results on MEWA's meta-training distribution for all the baseline algorithms. All their task policies converge (Fig. 11), together with PEARL's, VariBAD's, and DREAM's encoders (Fig. 12), and DREAM's exploration policy (Fig. 13). Note that MAML does not have an encoder, and DREAM is the only baseline algorithm with a separate exploration policy.

5.2.2 Meta-Testing Performance on MEWA

We now evaluate LaSER's ability to solve MEWA's meta-testing tasks. For each algorithm, we meta-test the agent obtained at the end of meta-training. Note that before being evaluated on a task, each agent is allowed to explore for $K = 4$ episodes. We report average returns over 10 seeds in Fig. 6 and Table 1. Because of the inherent randomness in MEWA's tasks, we take an additional step to ensure the results are not due to random chance. For each seed, we repeat the entire meta-test (i.e., both task exploration and exploitation) 9 times per agent for each of the 12 meta-testing tasks. For each repetition, we collect 20 exploitation episodes, using the same context vector, and then average over their returns. In addition to the results averaged over all tasks, we report per-task performance in Fig. 14.

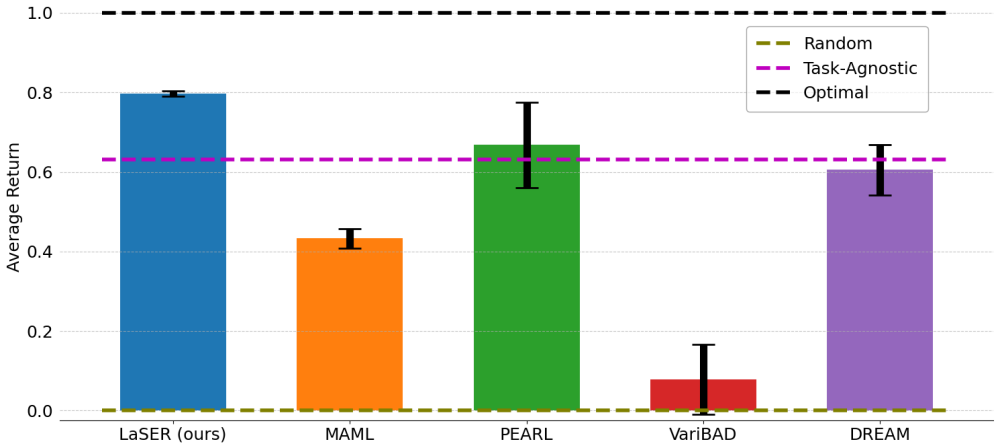

Figure 6: Average returns achieved on MEWA's meta-testing tasks. Each agent is meta-tested with an exploration budget of $K = 4$. Results are averaged over 10 seeds, with error bars indicating standard error. Note that we normalize results between the *random* and *optimal* baselines.

LaSER (ours)	MAML	PEARL	VariBAD	DREAM
$\underline{0.8} \pm \underline{0.021}$	0.43 ± 0.072	0.67 ± 0.32	0.08 ± 0.262	0.6 ± 0.191

Table 1: MEWA returns averaged across 10 seeds ($\pm$ std).

LaSER achieves the highest average returns among all meta-RL algorithms. It also proves to be the most stable algorithm, with a post-adaptation standard deviation of just 0.02 across all seeds. LaSER is one of only two methods to outperform the *task-agnostic* baseline. The other algorithms only outperform the *random* baseline. PEARL learns a strong global policy, but is less stable and underperforms compared to LaSER. While DREAM appears more stable, its performance is slightly below the *task-agnostic* baseline. In contrast, MAML and VariBAD perform poorly, with VariBAD being both unstable and close in performance to the *random* baseline.

5.2.3 Meta-Testing Performance on Meta-World and HopperMass

Fig. 7 (left) shows that LaSER achieves competitive performance on Meta-World ML10. For a meta-training budget of 50 million timesteps, we present the average success rate on the ML10 meta-testing tasks, measured at intervals of $4 \cdot 10^5$ timesteps. Note that for LaSER, the encoder and exploration policy have been pre-trained for N_{explore} iterations before starting the task policy training.

LaSER appears to be more sample efficient initially, outperforming VariBAD for approximately 20 million timesteps. For example, LaSER requires only 8 million timesteps to pass the 0.05 success rate threshold, i.e., 1.75 times fewer than VariBAD's 14 million. While VariBAD learns faster in the second half, both algorithms reach the 0.1 threshold after approximately 40 million timesteps and eventually achieve maximum

performances of 0.113 (LaSER) and 0.116 (VariBAD). LaSER's performance, however, degrades in the last 10 million timesteps, while VariBAD's remains stable.

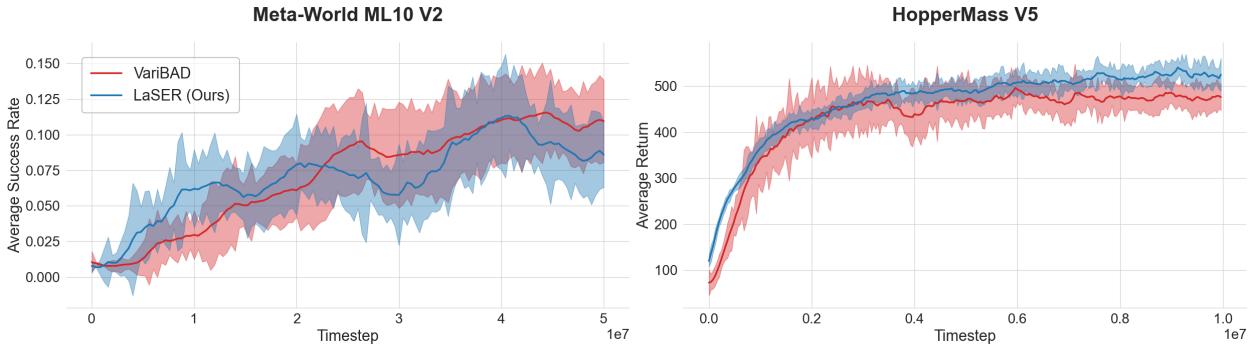

Figure 7: Meta-testing performance, averaged across 10 random seeds (± standard error). We use EMA smoothing with $\alpha_{\text{EMA}} = 0.1$. (left) Average success rate on Meta-World ML10 meta-testing tasks for an exploration budget of $K = 2$. We measure performance every $4 \cdot 10^5$ timesteps. (right) Average return on HopperMass OOD tasks for $K = 4$, measured at intervals of $40,000$ thousand timesteps.

As shown in Fig. 7 (right), LaSER outperforms VariBAD in HopperMass in terms of OOD performance. We meta-train each agent for 10 million timesteps and measure OOD performance at intervals of $40,000$ timesteps. As before, LaSER's encoder and exploration policy are first pre-trained for N_{explore} iterations.

LaSER is more stable and adaptive to the HopperMass OOD tasks than VariBAD. Specifically, LaSER achieves a maximum average performance of 540, while VariBAD stops improving at 496. Additionally, LaSER's superior stability is shown in the lower average standard error (26.3) compared to VariBAD (30.3) during the final million timesteps.

5.3 Ablation Study: Task Solving

As an ablation study, we evaluate our approach for meta-training in-context task policies, which was introduced in Sec. 3.1. The results in this section correspond solely to the task-solving phase of Fig. 1. We meta-test task policies by rolling them out before and after they receive a task context $\boldsymbol{c}$. Our primary objective is to assess adaptability, i.e., a policy's ability to use $\boldsymbol{c}$ to improve its performance. We quantify this as the difference between post-adaptation and pre-adaptation return.

We make two important simplifications to the standard MEWA benchmark.

- To ensure a fair comparison of task policies alone, we meta-train and meta-test without task exploration or task learning. Instead, for each task, we use a ground truth, oracle-provided context vector $\boldsymbol{c}$, which is normally unavailable to the agent.
- To assess adaptability to non-stationary dynamics, we use a simpler, multi-task objective: we perform both meta-training and meta-testing on MEWA's 12 meta-testing tasks.

Despite these restrictions, MEWA's guarantee for the nonexistence of a globally optimal policy still holds. That is, only adaptive policies can achieve maximum return.

LaSER combines PPO and the proposed meta-reward r_t^+ (see Eq. 2) to optimize the in-context policy π_θ. We compare it to the simpler approach of using in-context policies optimized through standard PPO, as well as to Decision Adapters (Beukman et al., 2024, DAs), which use hypernetworks to generate task-specific policies.[5] For a fair comparison, the in-context PPO policy has the same architecture as LaSER's policy.

[5]Since Beukman et al. (2024) build and evaluate their DAs with ground-truth contexts in mind, their algorithm is agnostic to the context-learning mechanism, and, as reported by the authors, sensitive to noisy contexts. We therefore only consider DAs as baselines for LaSER's task policy, and not for the entire LaSER algorithm.

We also use PPO to optimize DAs, so the main difference between these and the other two methods is the architectural change. Appendix F.5 provides implementation details and hyperparameters for DAs.

The meta-training results are shown in Fig. 8 and Table 2. Our method achieves the highest average return while also stabilizing meta-training. It is also the only method to find a policy that adapts optimally, which occurs in 5 out of 10 seeds. In-context PPO is slightly less stable, presumably because the agent is trying to find a single policy that maximizes returns across all tasks. However, without task-specific adaptation, it fails to find such a globally optimal policy. Hypernetworks, on the other hand, are highly unstable. This is likely due to hypernetworks being inherently challenging to train (Ortiz et al., 2023; Chauhan et al., 2024; Beukman et al., 2024). They also achieve the lowest average return.

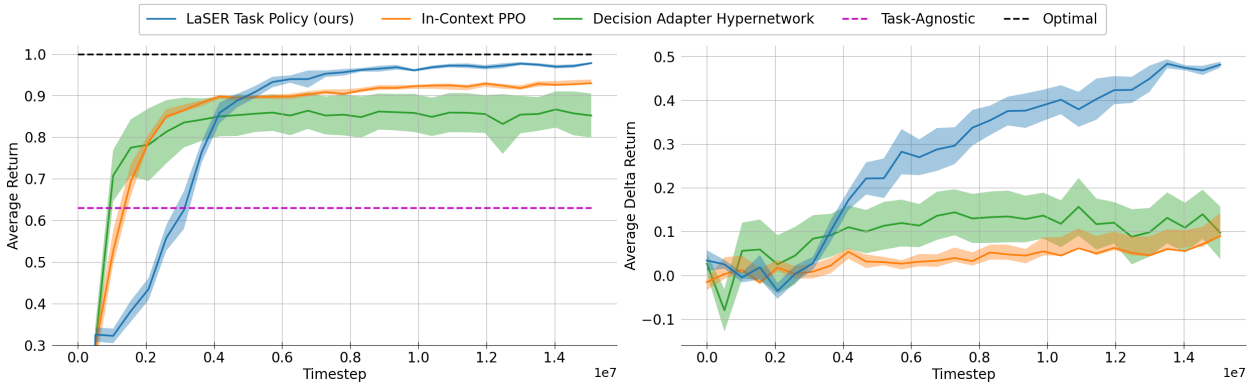

Figure 8: Task policy meta-training. We use oracle-provided task contexts $\boldsymbol{c}$, and meta-train and meta-test on the same set of 12 MEWA tasks. The shaded areas show the standard error of the average return across 10 random seeds. (left) Post-adaptation return of task policy π_θ, given task context $\boldsymbol{c}$. All returns are normalized between MEWA's *random* and *optimal* baselines. To improve readability, we restrict the plot to the $[0.3, 1.0]$ 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 $\boldsymbol{c}$.

	LaSER Task Policy (ours)	**In-Context PPO**	**Decision Adapter Hypernetwork**
Return	$\underline{0.97} \pm \underline{0.006}$	0.93 ± 0.014	0.86 ± 0.149
Delta Return	$\underline{0.47} \pm \underline{0.019}$	0.07 ± 0.067	0.12 ± 0.164

Table 2: Average returns and delta returns over the last $1e6$ timesteps, averaged across 10 seeds ($\pm$ std). We meta-train and meta-test on the same 12 MEWA tasks, and use oracle-provided context vectors $\boldsymbol{c}$.

The LaSER task policy also achieves greater task-adaptation success than the other two methods. After the meta-exploration phase, the agent stabilizes towards the end of meta-training. The task-conditioned policy then consistently outperforms the pre-adaptation policy. Hypernetwork-generated policies adapt better than in-context PPO but are also more unstable. Additionally, their low average return limits the benefits of this adaptability. In-context PPO does not explicitly optimize for adaptation, so its adaptability is approximately five times lower than our method.

Note that, at the beginning, our method learns more slowly than in-context PPO and hypernetworks. That is, it requires more samples to pass the average return thresholds of 0.9 (for PPO) and 0.85 (for hypernetworks). We attribute this to additional meta-exploration in the state space, performed in the context of the task space. The meta-reward r_t^+ encourages the policy to learn how to take in-context actions that increase the gap between $V(s_t)$ and $V(s_t, \boldsymbol{c})$, for a state s_t and context $\boldsymbol{c}$. This is in addition to the meta-exploration required to only maximize the standard RL objective $V(s_t)$. We hypothesize that in-context PPO and hypernetworks cannot surpass our method due to their lack of explicit task-space meta-exploration. The policies they produce optimize standard RL objectives, so they meta-explore accordingly.

Finally, we observe that our method is nearly as computationally efficient as in-context PPO and much faster than hypernetworks. We measure the average wall-clock time to roll out and optimize π_θ, per iteration. On average, hypernetwork-based agents are approximately 3.3× slower than ours, while our method is only 1.3× slower than in-context PPO. See Fig. 16 for a detailed comparison. We discuss how meta-rewards cause this additional overhead in Appendix A.2.

5.4 Ablation Study: 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 task contexts computed by LaSER, PEARL, VariBAD, and DREAM for each of the 12 meta-testing tasks in MEWA. 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 task context vector $\boldsymbol{c}$ using the agent's meta-trained encoder. Similarly, DREAM collects data using its exploration policy. We visualize two-dimensional projections of task contexts, computed using t-SNE (Van der Maaten & Hinton, 2008), in Fig. 9. The 12 tasks are sorted by similarity, i.e., average mistake probability, as in Table 3.

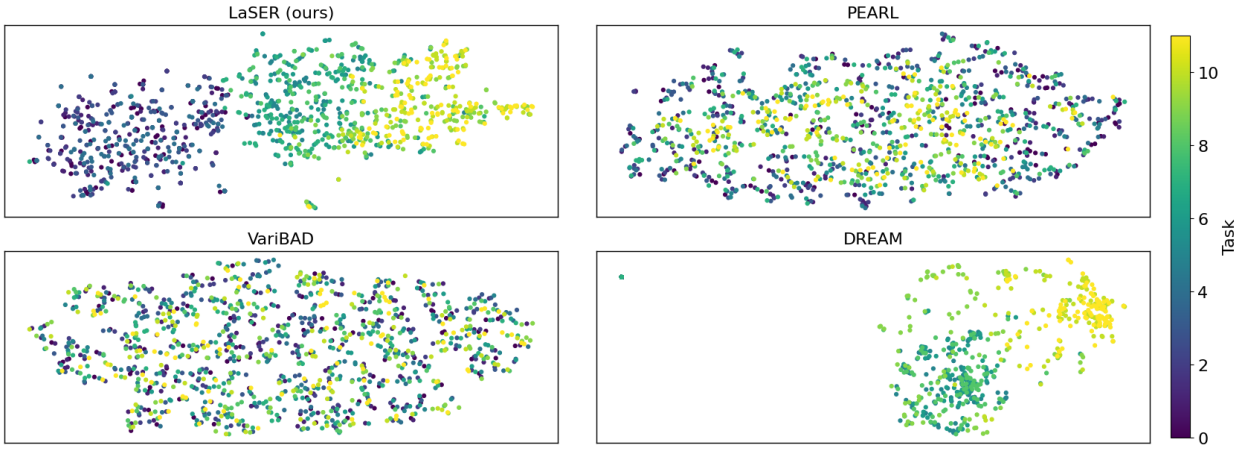

Figure 9: Latent representations of meta-episodes collected and encoded by meta-trained agents. Each of the 1200 points is a context vector belonging to one of the 12 meta-testing tasks in MEWA. The two-dimensional visualizations are computed using t-SNE. Different tasks have different colors.

For LaSER, we can find clusters of task contexts. Specifically, meta-episodes collected from the first five tasks, the next four tasks, and the final three tasks appear to be projected apart from each other. We note that the tasks in each cluster share common traits. In the first cluster, mistakes of type $x \in \{3, 4\}$ never occur. In the third cluster, $\boldsymbol{p}_x^{(i)} \geq 0.9$ for any $x \neq 4$. The second cluster's average mistake probabilities lie between those of the other two clusters. The separation appears to follow this trend. More details on the tasks' properties are shown in Table 3.

In contrast, PEARL's and VariBAD's latent contexts lack clear clustering that could distinguish data collected from different tasks. This low separation may make it more difficult to identify task-specific features. DREAM, on the other hand, appears to follow a similar three-cluster separation as LaSER. A distinguishable feature is that data collected from any of the first five tasks collapses into a single context representation.

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 of K can make it difficult to separate contexts of similar tasks.

6 Conclusion

We introduced LaSER, a new approach for meta-learning RL exploration. Our results demonstrate that LaSER outperforms previous meta-RL algorithms on the MEWA benchmark and is competitive on the Meta-World and HopperMass benchmarks. They also show that LaSER can meta-learn better task clustering during exploration. Additionally, we propose a novel meta-exploration bonus for training task policies efficiently. This outperforms previous approaches of meta-training in-context policies in both accumulated rewards and adaptability. Since our task-solving method is agnostic to how tasks are explored or represented, it can be integrated into any in-context meta-RL algorithm and then optimized using standard RL.

An important strength of LaSER is that, in the setting where contexts are meta-learned, it outperforms other algorithms, despite none of the methods being optimally adaptive. We argue that this is a result of meta-training with higher-quality task contexts. Note that the main role of these contexts remains 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 in future work.

LaSER is built upon the unique properties and requirements of the meta-RL framework, which we approach through the lens of few-shot adaptation. As a result, we tackle challenges that are specific to meta-RL, i.e., challenges that might not exist in the broader fields of RL or meta-learning. For example, our exploration algorithm leverages a key assumption about the structure of data collected in few-shot RL environments. Similarly, LaSER's task policy is meta-trained with the explicit goal that in-context policies must outperform context-agnostic policies in a meta-RL setting. This idea, realized as a form of extended meta-exploration over the task context space, leads to almost-optimal adaptive policies in MEWA, when ground-truth contexts are available. We hope our findings inspire future research to leverage the meta-RL framework in novel ways.

While LaSER outperforms other meta-RL algorithms, it still struggles to adapt to new tasks when task contexts are meta-learned. 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 instance, the encoder learned to separate MEWA's meta-testing tasks into three groups. This suggests that LaSER should be able to outperform its pre-adaptation policy in these tasks. Furthermore, our results show that the task policy can become more adaptive when restricting the meta-training task distribution and using ground truth contexts. We therefore propose the hypothesis that effectively combining these three components is non-trivial. Additionally, we suggest that future research should seek to explore and understand this issue.

Finally, we also note that LaSER ignores environmental reward maximization during task exploration.[6] On one hand, this keeps meta-RL agents general, discouraging them from overfitting to a single goal. On the other hand, this may cause issues when environmental rewards are necessary for the agent's survival. While we provide a simple method for combining our proposed exploration rewards with environmental rewards in Eq. 11, more complex approaches may be required to find the optimal trade-off between exploration and survival. We leave this interesting problem as future work.

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.

[6] LaSER still accounts for environmental rewards, as these are embedded in the context vector $\mathbf{\Gamma}$ used for guiding task exploration. However, the exploration objective does not explicitly encourage the maximization of such rewards.

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*, 2022.

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.

Nicolò Botteghi, Mannes Poel, and Christoph Brune. Unsupervised representation learning in deep reinforcement learning: A review. *IEEE Control Systems*, 45(2):26–68, 2025.

Vinod Kumar Chauhan, Jiandong Zhou, Ping Lu, Soheila Molaei, and David A Clifton. A brief review of hypernetworks in deep learning. *Artificial Intelligence Review*, 57(9):250, 2024.

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.

Akram Erraqabi, Marlos C Machado, Mingde Zhao, Sainbayar Sukhbaatar, Alessandro Lazaric, Denoyer Ludovic, and Yoshua Bengio. Temporal abstractions-augmented temporally contrastive learning: An alternative to the laplacian in rl. In *Uncertainty in Artificial Intelligence*, pp. 641–651. PMLR, 2022.

Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Russ R Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems*, 35:35603–35620, 2022.

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.

Jake Grigsby, Justin Sasek, Samyak Parajuli, Daniel Adebi, Amy Zhang, and Yuke Zhu. Amago-2: Breaking the multi-task barrier in meta-reinforcement learning with transformers. *Advances in Neural Information Processing Systems*, 37:87473–87508, 2024.

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.

Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Stenberg Hansen, Angelos Filos, Ethan Brooks, maxime gazeau, Himanshu Sahni, Satinder Singh, and Volodymyr Mnih. In-context reinforcement learning with algorithm distillation. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=hy0a5MMPUv`.

Jonathan Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. Supervised pretraining can learn in-context reinforcement learning. *Advances in Neural Information Processing Systems*, 36:43057–43083, 2023.

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.

Reginald McLean, Evangelos Chatzaroulas, Luc McCutcheon, Frank Röder, Tianhe Yu, Zhanpeng He, KR Zentner, Ryan Julian, JK Terry, Isaac Woungang, et al. Meta-world+: An improved, standardized, rl benchmark. *arXiv preprint arXiv:2505.11289*, 2025.

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.

Amir Moeini, Jiuqi Wang, Jacob Beck, Ethan Blaser, Shimon Whiteson, Rohan Chandra, and Shangtong Zhang. A survey of in-context reinforcement learning. *arXiv preprint arXiv:2502.07978*, 2025.

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.

Mohammadreza Nakhaeinezhadfard, Aidan Scannell, and Joni Pajarinen. Entropy regularized task representation learning for offline meta-reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 19616–19623, 2025.

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.

Jose Javier Gonzalez Ortiz, John Guttag, and Adrian Dalca. Magnitude invariant parametrizations improve hypernetwork learning. *arXiv preprint arXiv:2304.07645*, 2023.

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.

Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. Promp: Proximal meta-policy search. In *International Conference on Learning Representations*, 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.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012.

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

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30(1), 2016.

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*, 2016a.

Yuhui Wang, Hao He, and Xiaoyang Tan. Truly proximal policy optimization. In *Uncertainty in artificial intelligence*, pp. 113–122. PMLR, 2020.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pp. 1995–2003. PMLR, 2016b.

Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

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, 2022a.

Mengdi Xu, Yikang Shen, Shun Zhang, Yuchen Lu, Ding Zhao, Joshua Tenenbaum, and Chuang Gan. Prompting decision transformer for few-shot policy generalization. In *international conference on machine learning*, pp. 24631–24645. PMLR, 2022b.

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.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pp. 1094–1100. PMLR, 2020.

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

$$\mathcal{L}_t^{\text{PPO}}(\theta) = \mathbb{E}_t \left[\mathcal{L}_t^{\text{CLIP}}(\theta) - c_1 \mathcal{L}_t^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t, \boldsymbol{c}) \right], \tag{12}$$

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 $V_\theta(s_t, \boldsymbol{c})$, while $S[\pi_\theta]$ is the policy entropy. Note that the sole change from the original description of PPO is that $\pi_\theta(a_t \mid s_t, \boldsymbol{c})$ and $V_\theta(s_t, \boldsymbol{c})$ depend not only on the state s_t, but also on the task context $\boldsymbol{c}$. The clipped surrogate objective $\mathcal{L}_t^{\text{CLIP}}$ is defined as

$$\mathcal{L}_t^{\text{CLIP}} = \mathbb{E}_t \Big[\min \Big(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \hat{A}_t \Big) \Big], \tag{13}$$

for a constant ϵ, policy probability ratio $r_t(\theta)$, and estimated advantage $\hat{A}_t$. The probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t,\boldsymbol{c})}{\pi_{\theta_{\text{old}}}(a_t|s_t,\boldsymbol{c})}$ is computed using the parameters θ_{old} from before an update. Eq. 13 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 $\delta_t^V = r_t + \gamma V(s_{t+1}, \boldsymbol{c}) - V(s_t, \boldsymbol{c})$ 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 meta-reward r_t^+, PPO can be used to optimize π_θ to solve meta-RL tasks. That is, we compute the TD error

$$\delta_t^V(\theta) = r_t + \beta\, w(s_t, \boldsymbol{c})\, S[\pi_\theta](s_t, \boldsymbol{c}) + \gamma V(s_{t+1}, \boldsymbol{c}) - V(s_t, \boldsymbol{c}), \tag{14}$$

where $w(s_t, \boldsymbol{c}) = \max(0, \tanh(V(s_t) - V(s_t, \boldsymbol{c}) - \zeta))$. 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, $w(s_t, \boldsymbol{c})$ measures how effective the policy π_θ is at using the context $\boldsymbol{c}$, 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 context space is explored enough in a meta-RL setting. By introducing $w(s_t, \boldsymbol{c})$, whenever $\boldsymbol{c}$ does not lead to an improvement above a threshold ζ, the policy is urged to explore from state s_t, thus learning more about $\boldsymbol{c}$. As the agent improves at using $\boldsymbol{c}$ to maximize return, the exploration bonus $w(s_t, \boldsymbol{c})S[\pi_\theta](s_t, \boldsymbol{c})$ at state s_t decreases. The standard PPO objective is only recovered when $V(s_t) \leq V(s_t, \boldsymbol{c}) + \zeta$. This signals that the agent has learned how to use $\boldsymbol{c}$ 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) and 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 $\mathbf{B}'_j = h_{\text{rec}}(\boldsymbol{c}; \omega_{\text{rec}})$, where $\boldsymbol{c} = g_\omega(\mathbf{B}_j^{\text{masked}})$. 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 $\boldsymbol{c}$ 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 Implementation Details

We provide additional details on our implementation of the LaSER algorithm and our architecture. This section is complemented by the hyperparameters given in Appendix H.

Algorithm 3 train_encoder()

Input $p(\mathcal{M})$, task distribution;
$\pi_\phi^{\text{explore}}$, exploration policy; g_ω, encoder;
$\omega_{\text{rec}}, \omega_{\text{t-rec}}$, head parameters
Output $\omega, \omega_{\text{rec}}, \omega_{\text{t-rec}}$, updated parameters

$\mathcal{B} \leftarrow \left\{ \mathbf{B} \sim P_{\mathcal{M}_i}^{\pi_\phi^{\text{explore}}} \middle| \mathcal{M}_i \sim p(\mathcal{M}) \right\}$ ▷ Collect an ordered batch of data from multiple tasks
$\mathcal{B}^{\text{masked}} \leftarrow \{\text{mask}(\mathbf{B}) | \mathbf{B} \in \mathcal{B}\}$
for $u \in [N_{\text{encoder}}]$ **do**
  $\left(\boldsymbol{z}, \left\{\mathbf{Z}^{(1)}, \ldots, \mathbf{Z}^{(|\mathcal{B}|)}\right\}\right) \leftarrow f_b(\mathcal{B}^{\text{masked}}; \omega_b)$
  $\mathcal{L}_{\text{LaSER}} \leftarrow 0$
  for $i \in \left[|\mathcal{B}|\right]$ **do**
    $\mathbf{B} \leftarrow \mathcal{B}_i$; $\mathbf{B}^{\text{masked}} \leftarrow \mathcal{B}_i^{\text{masked}}$
    for $j \in [Q]$ **do**
      $\boldsymbol{\Gamma} \leftarrow f_u(\mathbf{B}_j^{\text{masked}}; \omega_u)$
      $\boldsymbol{c} \leftarrow h_{\omega_h}(\boldsymbol{z}, \mathbf{Z}_j^{(i)}, \boldsymbol{\Gamma})$ ▷ Eq. 9
      $\mathbf{B}'_j \leftarrow h_{\text{rec}}(\boldsymbol{c}; \omega_{\text{rec}})$ ▷ Compute reconstructed input
    end for
    $j \sim U(Q)$
    $\mathbf{C} \leftarrow h_{\text{t-rec}}(g_\omega(\mathbf{B}), g_\omega(\mathbf{B}_j); \omega_{\text{t-rec}})$
    $\delta_j \leftarrow 1 - \exp\left(-\xi\overline{(\mathbf{B}_j\mathbf{C} - \mathbf{B})^2}\right)$ ▷ Eq. 7
    $\omega \leftarrow \omega_u \oplus \omega_b \oplus \omega_h$
    $\mathcal{L}_{\text{LaSER}}^{(i)}(\omega, \omega_{\text{rec}}, \omega_{\text{t-rec}}) \leftarrow$ compute using $\mathbf{B}$, $\mathbf{B}'$, $\mathbf{C}$, and δ_j ▷ Eq. 6, 8, 10
    $\mathcal{L}_{\text{LaSER}} \leftarrow \mathcal{L}_{\text{LaSER}} + \frac{1}{|\mathcal{B}|}\mathcal{L}_{\text{LaSER}}^{(i)}(\omega, \omega_{\text{rec}}, \omega_{\text{t-rec}})$
  end for
  $\omega, \omega_{\text{rec}}, \omega_{\text{t-rec}} \leftarrow$ update using $\nabla\mathcal{L}_{\text{LaSER}}$
end for
return $\omega, \omega_{\text{rec}}, \omega_{\text{t-rec}}$

Algorithm 4 train_exploration_policy()

Input $p(\mathcal{M})$, task distribution; $\pi_\phi^{\text{explore}}$, exploration policy; g_ω, encoder
Output ϕ, updated parameters

$\mathcal{D} \leftarrow []$
for $\mathcal{M}_i \sim p(\mathcal{M})$ **do**
  $\mathcal{D}^{(K)} \leftarrow []$
  for $k \in [K]$ **do**
    for $t \in [H]$ **do**
      $a_t^k \sim \pi_\phi^{\text{explore}}\big(\cdot \mid s_t^k, \mathbf{\Gamma}^{(:k,:t-1)}\big)$
      $\mathcal{D}^{(k,t)} \leftarrow (s_t^k, a_t^k, r_t^k)$
      $\mathbf{\Gamma}^{(:k,:t)} \leftarrow f_u\left(\mathcal{D}^{(:k,:t)};\ \omega_u\right)$
      Collect s_{t+1}^k, r_{t+1}^k by taking action a_t^k in $\mathcal{M}_i$
    end for
  end for
  $\mathbf{\Gamma} \leftarrow f_u(\mathcal{D}^{(K)}; \omega_u)$; $\boldsymbol{d} \leftarrow \frac{1}{K} S_C(\mathbf{\Gamma}^\mathsf{T}, \mathbf{\Gamma})^\mathsf{T}\mathbf{1}$
  $\tilde{R}_k(s_t^k, a_t^k) \leftarrow$ compute for each $k \in [K], t \in [H]$ ▷ Eq. 4
  Replace environment rewards in $\mathcal{D}^{(K)}$ with the corresponding $\tilde{r}_t^k = \tilde{R}_k(s_t, a_t)$
  $\mathcal{D} \leftarrow [\mathcal{D}, \mathcal{D}^{(K)}]$
end for
for $u \in [N_{\text{PPO}}]$ **do**
  Optimize ϕ on transitions from $\mathcal{D}$ using PPO
end for
return ϕ

Algorithm 5 train_task_policy()

Input $p(\mathcal{M})$, task distribution;
  π_θ, task policy; $\pi_\phi^{\text{explore}}$, exploration policy; g_ω, encoder;
  $\hat{\boldsymbol{z}}$, shared component
Output θ, updated parameters

$\mathcal{B} \leftarrow []$
for $\mathcal{M}_i \sim p(\mathcal{M})$ **do**
  $\mathcal{D}^{(K)} \sim P_{\mathcal{M}_i}^{\pi_\phi^{\text{explore}}}$ ▷ Sample exploration meta-episode
  $\boldsymbol{c} \leftarrow g_\omega(\mathcal{D}^{(K)})$ ▷ Compute $\boldsymbol{c}$, using $\hat{\boldsymbol{z}}$ as the shared component
  $\mathcal{B} \leftarrow [\mathcal{B}, \tau \sim P_{\mathcal{M}_i}^{\pi_\theta(a|s,\boldsymbol{c})}]$ ▷ Sample exploitation episode
end for
for $u \in [N_{\text{PPO}}]$ **do**
  $\hat{A}_\theta \leftarrow []$
  for $\tau \in \mathcal{B}$ **do**
    for $t \in [H]$ **do**
      Compute $V(s_t, \boldsymbol{c})$, $V(s_t)$, and $S[\pi_\theta](s_t, \boldsymbol{c})$
      $w(s_t, \boldsymbol{c}) \leftarrow \max\left(0, \tanh\left(V(s_t) - V(s_t, \boldsymbol{c}) - \zeta\right)\right)$ ▷ Eq. 3
      $r_t^+ \leftarrow r_t + \beta w(s_t, \boldsymbol{c}) S[\pi_\theta](s_t, \boldsymbol{c})$ ▷ Eq. 2
      $\hat{A}_t^{\text{GAE}}(\theta) \leftarrow$ compute using r_t^+
      $\hat{A}_\theta \leftarrow \left[\hat{A}_\theta, \hat{A}_t^{\text{GAE}}(\theta)\right]$
    end for
  end for
  Optimize θ on transitions from $\mathcal{B}$ with advantages $\hat{A}_\theta$ using PPO
end for
return θ

D.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` activation functions (Hendrycks & Gimpel, 2016). 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 the transformer in f_u is passed through a single-layer feed-forward network that outputs $\boldsymbol{\Gamma}$. Similarly, the transformer output of f_b is independently processed by two separate single-layer feed-forward networks, outputting $\boldsymbol{z}$ and $\mathbf{Z}$. Each of these three networks has size 64 and uses `tanh` activation functions.

When using data from multiple tasks, $\boldsymbol{\Gamma}$ and $\mathbf{Z}$ are computed separately for each task. In contrast, $\boldsymbol{z}$ is computed with data from multiple tasks, leading to a general representation of the entire distribution.

Before computing $\boldsymbol{z}$, we apply average pooling to reduce the dimensionality of the output of f_b. We treat $\mathbf{Z}$ as a batch of Q meta-episode representations, such that each slice $\mathbf{Z}_j$, for $j \in [Q]$, is computed independently by the feed-forward network. Similarly, each of the Q meta-episodes has its own representation $\boldsymbol{\Gamma}$, computed independently by the feed-forward network in f_u.

The encoder g_ω has two final transformer layers, h_{ω_h}, similar to those in f_b. The first layer combines $\mathbf{Z}$ and $\boldsymbol{\Gamma}$. The second computes $\boldsymbol{c}$ by combining the output of the first with $\boldsymbol{z}$. The reconstruction head h_{rec} is a single linear layer that transforms each of the HK vectors in $\boldsymbol{c}$ from d_{model} to d, thus computing the reconstructed input. The task-reconstruction head $h_{\text{t-rec}}$ is a $(256, 128, 128)$ feed-forward network with `GELU` activations, used for computing the tensor of coefficients $\mathbf{C}$. We have found, empirically, that using only the task-specific representations $\mathbf{Z}$ as input for $h_{\text{t-rec}}$ is enough. That is, for a given $j \in [Q]$, we compute

$$\begin{aligned} \mathbf{C} &= h_{\text{t-rec}}(\mathbf{Z}, \mathbf{Z}_j; \omega_{\text{t-rec}}), \\ \text{where } (\cdot, \mathbf{Z}) &= f_b(\mathbf{B}; \omega_b), \end{aligned}$$

and optimize $\omega_{\text{t-rec}}$ such that $\mathbf{C}$ approximates the linear transformation that maps $\mathbf{B}_j$ to $\mathbf{B}$. To train g_ω, h_{rec}, and $h_{\text{t-rec}}$, we use the Adam optimizer (Kingma & Ba, 2015).

D.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 $\boldsymbol{\Gamma}^{(:k,:t)}$ as input, which are computed by a shared, single-layer, `tanh`-activated network.

We use similar architectures for π_θ and V_θ. However, $\boldsymbol{c}$ is first mapped to a lower-dimensional representation by a feed-forward context encoder with `tanh` activations. The same encoder is used for both π_θ and V_θ. The hyperparameters of π_θ and V_θ are given in Table 8.

To optimize both policies, each PPO update is run for N_{PPO} epochs, using minibatches. The exploration policy is always meta-trained using $N_{\text{PPO}} = 4$ with 2 minibatches. For the task policy, these two hyperparameters are environment-dependent and shown in Table 8. Moreover, we normalize the state s, the reward r, and the context $\boldsymbol{c}$ before passing them to the task policy. We use the Adam optimizer, and clip the norm of the gradients to 0.5 before updating parameters ϕ and θ. Additionally, we set the coefficients in Eq. 12 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, weighted by c_{PFO}, which is set based on the environment. 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.

D.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 $\pi_\phi^{\text{explore}}$ in different tasks. Each element $\mathbf{B} \in \mathcal{B}$ is a tensor that contains Q meta-episodes from a task $\mathcal{M}_i \sim p(\mathcal{M})$.

We update g_ω on the masked dataset $\mathcal{B}^{\text{masked}}$ for N_{encoder} epochs (see Alg. 3). To improve meta-training sample efficiency, we use all Q meta-episodes in each tensor $\mathbf{B}$ to compute $\mathcal{L}_{\text{rec}}$ and $\mathcal{R}$. In practice, each $\mathcal{B}$ is stored in a buffer and reused in future updates. 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 losses $\mathcal{L}_{\text{rec}}$ and $\mathcal{L}_{\text{t-rec}}$ in Eq. 10, 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.

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 pretrained 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.

In practice, LaSER's encoder g_ω and exploration policy $\pi_\phi^{\text{explore}}$ are meta-trained using an Nvidia A100 80GB GPU. In the second phase, we switch to an Nvidia RTX 4070 Ti SUPER 16GB GPU to meta-train the task policy π_θ. All environment data is collected using CPU computations only.

E Benchmarks Details

This section provides additional implementation details for the meta-RL benchmarks used in this work.

E.1 MEWA

We continue the discussion from Sec. 5.1.1 and provide more details on the tasks available in the MEWA benchmark. In our implementation, we use the publicly available official code for MEWA, at https://github.com/RStoican/MEWA.

In a critical state s_x of type x, a risky action a_{risk} can either lead to a state s' or s'_x. We use s' to denote that a mistake has been avoided and the agent has progressed in the task. In contrast, s'_x denotes that a mistake has happened, resulting in a large delay in task completion and thus, lower returns. The probability of a_{risk} leading to a mistake depends on both the mistake's type x and the dynamics of task $\mathcal{M}_i \sim p(\mathcal{M})$. Formally, we define $T_i(s'_x \mid s_x, a_{\text{risk}}, \mathcal{M}_i) = \boldsymbol{p}_x^{(i)}$ to be the task-specific transition leading to a mistake. Here, the vector $\boldsymbol{p}^{(i)}$, with $\boldsymbol{p}_y^{(i)} \leq \boldsymbol{p}_x^{(i)}$ for all $y < x$, denotes the probabilities of making a mistake of each type. The probability of avoiding a mistake is then simply $T_i(s' \mid s_x, a_{\text{risk}}, \mathcal{M}_i) = 1 - \boldsymbol{p}_x^{(i)}$.

An agent's second option is to take a safe action a_{safe} in a critical state s_x. This comes at the cost of a small delay, and leads to a critical state s_{x-1}, with $\boldsymbol{p}_{x-1}^{(i)} \leq \boldsymbol{p}_x^{(i)}$. Formally, the dynamics are given by the task-agnostic transition $T_i(s_{x-1} \mid s_x, a_{\text{safe}}, \mathcal{M}_i) = 1$.

Table 3 provides the 12 meta-testing tasks used for the results in Sec. 5. We describe each task by its corresponding vector of mistake probabilities $\boldsymbol{p}^{(i)} \in [0, 1]^4$. Additionally, for each task $\mathcal{M}_i$, we provide the average probability of a mistake of any type happening, computed as $1/4 \sum_{x=1}^{4} \boldsymbol{p}_x^{(i)}$. As previously mentioned, we use this as a rough measure of similarity between tasks. Finally, we mark tasks that are outside of MEWA's meta-training distribution (i.e., cannot be sampled during meta-training) as OOD tasks.

Task Index	Mistake Probability				Average Mistake Probability	OOD
	Type I	Type II	Type III	Type IV		
0	0	0	0	0	0.000	No
1	0.05	0	0	0	0.013	No
2	0.1	0	0	0	0.025	No
3	0.772	0	0	0	0.193	No
4	0.672	0.618	0	0	0.322	No
5	0.622	0.618	0.577	0.434	0.563	No
6	0.622	0.618	0.577	0.534	0.588	Yes
7	0.872	0.818	0.777	0	0.617	Yes
8	0.672	0.668	0.627	0.584	0.638	Yes
9	0.972	0.968	0.927	0.384	0.813	Yes
10	0.972	0.968	0.927	0.784	0.913	Yes
11	0.972	0.968	0.927	0.884	0.938	Yes

Table 3: The configuration of the 12 MEWA tasks used for meta-testing agents. A task is described by a 4-dimensional vector. Each dimension denotes 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. Finally, tasks marked as OOD are outside MEWA's meta-training task distribution.

E.2 Meta-World

Meta-World was proposed by Yu et al. (2020). Its latest version, which was introduced by McLean et al. (2025), offers a unified and consistent framework for easily evaluating and comparing meta-RL algorithms. The most important change in this version is the option of selecting between Meta-World's initial set of task-specific reward functions (V1) and the newer set (V2). To be more consistent with the recent literature, we choose to follow V2 in our work. All Meta-World experiments are performed on the ML10 evaluation protocol. Ten tasks, with randomized positions for objects and goals, are selected for meta-training. Five structurally similar novel tasks are then used for meta-testing (for more details, see Yu et al. (2020)). To implement Meta-World, we use the official open codebase provided at https://github.com/Farama-Foundation/Metaworld.

E.3 HopperMass

Our implementation of MuJoCo HopperMass is similar to the one provided at https://github.com/MohammadrezaNakhaei/ER-TRL by Nakhaeinezhadfard et al. (2025). The main difference from other implementations used in the literature is the focus on generating OOD tasks during meta-testing. The hopper model is characterized by four vectors: body mass, body inertia, friction, and degree of freedom damping. When generating a new task, each element from each of these vectors is scaled by a different weight. Weights are sampled from $[1.5^{-1}, 1.5^{1}]$ for meta-training tasks and from $[1.5^{-1.5}, 1.5^{-1}) \cup (1.5^{1}, 1.5^{1.5}]$ for OOD meta-testing tasks.

F Baseline Algorithms Details

We provide details for the architectures, hyperparameters, and the meta-training process for the baseline algorithms used in Sec. 5. 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. All baseline algorithms are meta-trained using an Nvidia RTX 4070 Ti SUPER 16GB GPU.

F.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.

F.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 contexts $\boldsymbol{c}$. 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 by conditioning its policy on a context $\boldsymbol{c}$ sampled from a prior distribution over tasks. It additionally collects 600 more timesteps using a $\boldsymbol{c}$ 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 context vectors $\boldsymbol{c}$. The task policy is a $(128, 128, 128)$ feed-forward network with `tanh` activations.

F.3 VariBAD

We use the code at https://github.com/lmzintgraf/varibad for VariBAD (Zintgraf et al., 2021b). We meta-train for 2350 (MEWA), 18750 (Meta-World), or 3125 (HopperMass) iterations. In MEWA, we meta-train on 3000 tasks. For any benchmark, at each iteration, we collect 200 timesteps per task from 16 different tasks. These timesteps are stored in a buffer of 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 (Kingma & Welling, 2014, VAE) to encode data collected from tasks into a latent context $\boldsymbol{c}$. In MEWA, 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 contexts $\boldsymbol{c}$. The decoder takes $\boldsymbol{c}$ 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 preprocessed 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 for MEWA as LaSER's task policy (see Appendix D.2 and Table 8). However, we do not use the additional PFO objective. Finally, before passing s and $\boldsymbol{c}$ to the policy, we embed them into 64-dimensional representations using separate single-layer networks with `tanh` activations. Similarly to LaSER, all experiments use the discount factor $\gamma = 0.99$.

For Meta-World and HopperMass, we use the hyperparameters proposed by Beck et al. (2023a) for ML10 as a starting point for our tuning. In Meta-World, we use 2 PPO epochs with 8 minibatches with a learning rate of $3 \cdot 10^{-4}$. In HopperMass, we use batches of 2 episodes to update the VAE. For the task policy, we have 4 PPO updates with 8 minibatches and a learning rate of $3 \cdot 10^{-4}$. The VAE for both Meta-World and HopperMass computes 10-dimensional latent context vectors $\boldsymbol{c}$, which are used by a $(256, 256, 128)$ task policy. Moreover, we normalize the state s, the reward r, and the context $\boldsymbol{c}$ before passing them to the task policy. The rest of the hyperparameters are the same as for the VariBAD models meta-trained on MEWA.

F.4 DREAM

We use the public code provided by Liu et al. (2021) at https://github.com/ezliu/dream to implement DREAM. We meta-train for 100000 iterations. At each iteration, the model first collects $K = 4$ exploration episodes and updates the exploration policy. Afterwards, the task policy, conditioned on the exploration meta-episode, is used to collect an exploitation episode, and then updated.

The decoder used by DREAM takes a history of transitions of type (s, a, r, s') as input. Each transition is individually embedded, using layers of size 64 for states, 16 for rewards, and 16 for actions. For each transition, we then concatenate these embeddings and apply a linear layer of size 64. Then, an LSTM with hidden dimension 128 encodes the history of embedded transitions. Finally, the output of this LSTM is postprocessed by a $(128, 64)$ feed-forward network with `ReLU` activations.

Both the task and exploration policies are trained with dueling double Q-learning (Wang et al., 2016b; Van Hasselt et al., 2016), parameterised as deep recurrent Q-networks. The deep Q-network (DQN) used to train the task policy takes the current state, history, and task context as inputs. The state and task context are separately embedded using matrices of size 64. The history is encoded using a process similar to the one used by the encoder. However, following Liu et al. (2021), the history is composed of timesteps instead of transitions (i.e., no next state s'), rewards are not embedded, actions use a 32-dimensional linear layer, and the hidden size of the LSTM is decreased to 64. A feed-forward network of size $(256, 64)$ with `ReLU` activations is then applied to the concatenation of these output embeddings. The task DQN has two final linear heads for computing the value and advantage functions. The exploration policy's DQN has a similar architecture.

Each of the two DQNs has its own replay buffer, each storing up to 16000 sequences of 50 timesteps. For each sequence, the timesteps are stored in consecutive order. We train both DQNs using the Adam optimizer with a learning rate of 1×10^{-4}. At each iteration, we update the DQNs 4 times using batches of size 32, clipping the norm of the gradients to 10. The DQNs are then synchronized after every 5000-th update. Finally, in all experiments, we use the discount factor $\gamma = 0.99$.

F.5 Decision Adapters

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 D.2). It is also meta-trained with the same hyperparameters (see Table 8). 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 context $\boldsymbol{c}$ as input. Following Beukman et al. (2024), we also use a skip connection between the input and output of this network with hypernetwork-generated weights.

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 $\boldsymbol{c}$, we pre-process it into

a 4-dimensional representation using a single `tanh`-activated layer. Finally, the actor and the critic have separate hypernetworks.

G Additional Results

We provide results that complement those in Sec. 5.

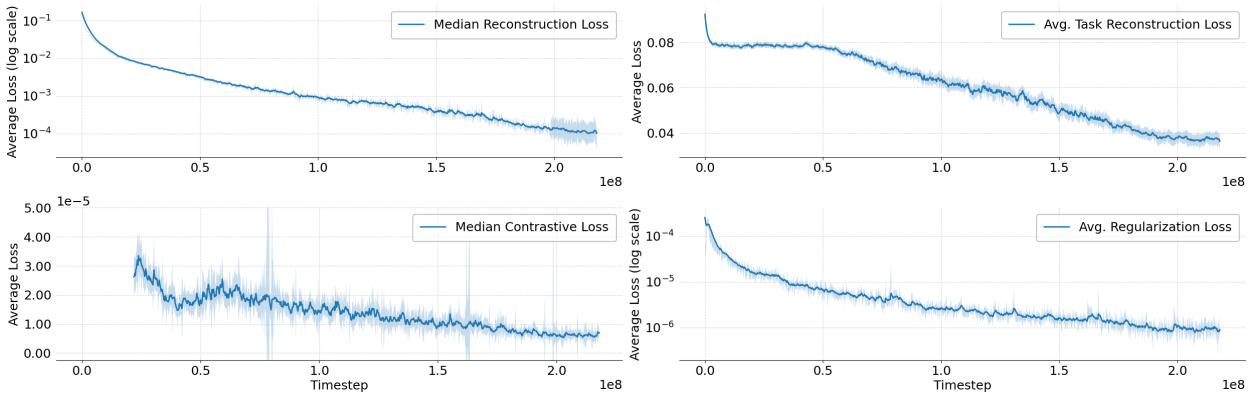

Figure 10: Encoder loss for LaSER, computed on tasks from MEWA's meta-training distribution. We show separate plots for each term in the encoder loss function. For 10 random seeds, we provide the standard error as the shaded areas, average the second and fourth metrics, and report the median for the other two. We found the median to be more appropriate, as one of the seeds was unstable towards the end of meta-training, for approximately $0.2e8$ timesteps. For this same reason, we restrict the plot of the contrastive loss to the range $[-2.5\text{e}{-6}, 5\text{e}{-5}]$. To aid readability for exponential losses, the reconstruction and regularization losses use a logarithmic scale. All metrics are smoothed using EMA with $\alpha_{\text{EMA}} = 0.2$.

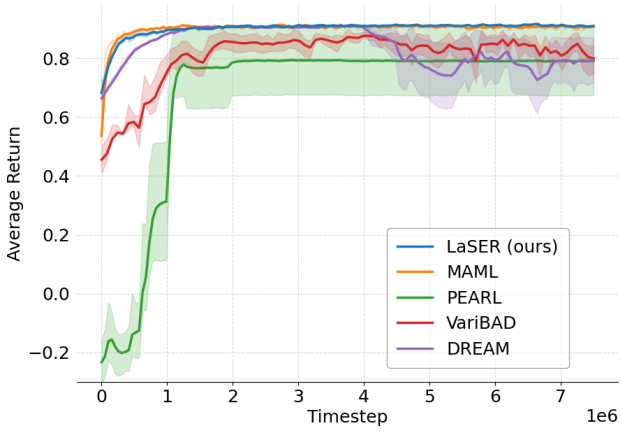

Figure 11: Undiscounted returns of task-solving policies on tasks from MEWA's meta-training task distribution. The shaded areas report the standard error of the average return across 10 seeds. We use EMA smoothing with $\alpha_{\text{EMA}} = 0.6$.

H Hyperparameters

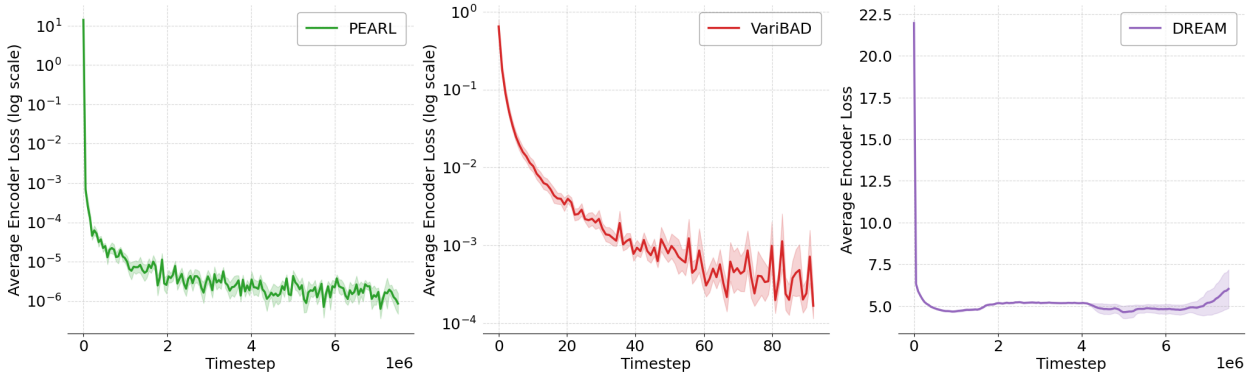

Figure 12: Encoder loss for PEARL, VariBAD, and DREAM on tasks from MEWA's meta-training task distribution. The shaded areas report the standard error of the average loss across 10 seeds. Since PEARL's and VairBAD's losses decay exponentially, we use logarithmic scales.

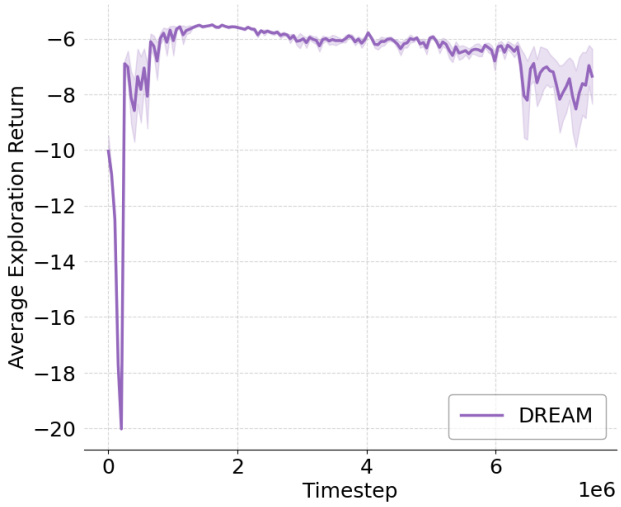

Figure 13: Exploration return for DREAM's exploration policy. Shaded areas are the standard error of the average over 10 random seeds.

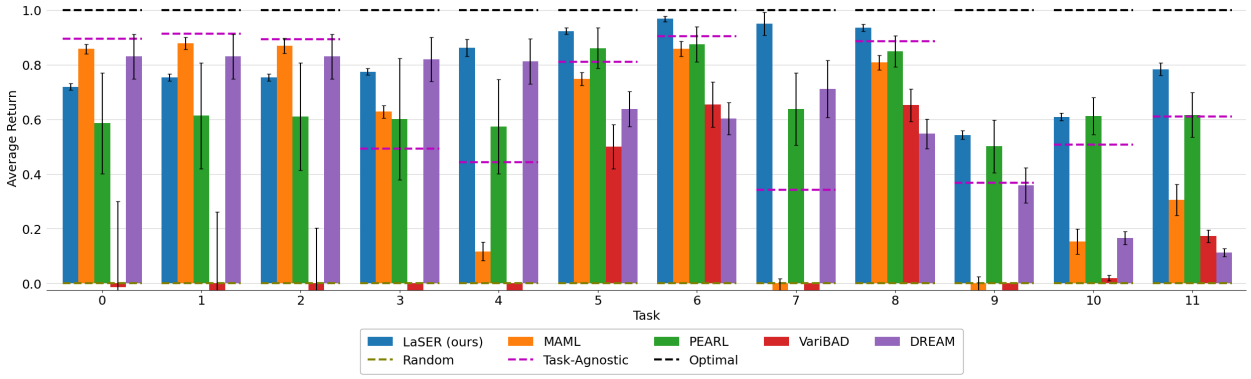

Figure 14: Average per-task returns for each of the 12 meta-testing tasks in MEWA. Each agent is meta-tested with an exploration budget of $K = 4$. Results are averaged over 10 seeds, with error bars indicating standard error across seeds. The three baselines are computed on a per-task basis. We normalize returns between the *random* and *optimal* baselines. For better visualization, we also restrict the plot to the $[-0.02, 1.0]$ range.

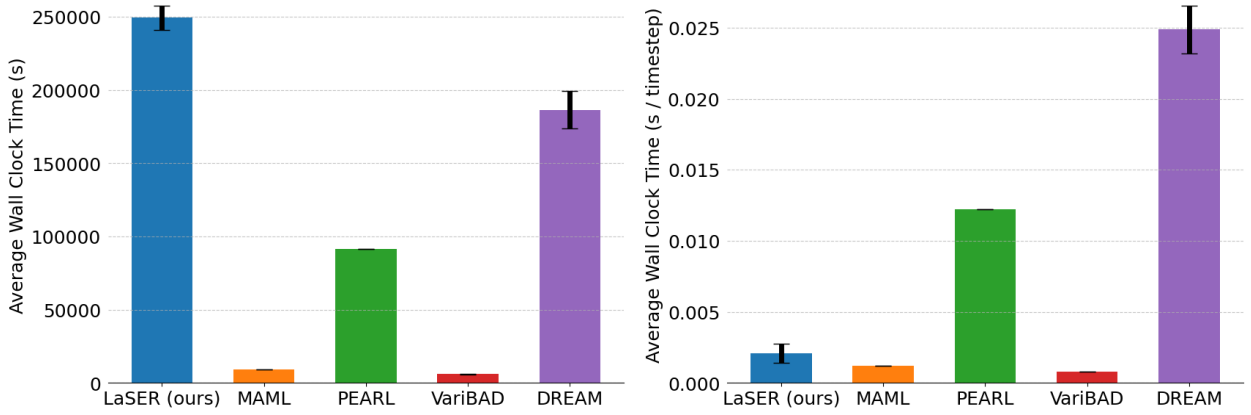

Figure 15: Wall clock time required to meta-train each method. We report the standard error of the mean over 10 seeds. (Left) Total meta-training time. (Right) Meta-training time averaged over all timesteps.

Algorithm	Parameters		Component Breakdown	
	Meta-Training	Meta-Testing	Component	Parameters
LaSER (ours)	78,394,281	8,111,314	Encoder g_ω	4,484,672
			Encoder Heads $h_{\text{rec}}, h_{\text{t-rec}}$	70,282,967
			Exploration Policy $\pi_\phi^{\text{explore}}$	109,193
			Task Policy (No Context Extractor) π_θ	141,961
			Task Policy Context Extractor	3,375,488
MAML	5,676	5,676	Task Policy	5,676
PEARL	233,271	233,271	Encoder	87,210
			Task Policy	146,061
VariBAD	186,580	178,243	Encoder	76,538
			Decoder	8,337
			Task Policy (No Context Extractor)	101,001
			Task Policy Context Extractor	704
DREAM	1,080,546	1,080,546	Encoder	426,880
			Exploration Policy	214,809
			Task Policy	438,857

Table 4: Number of parameters for each algorithm. All architectures are for the MEWA benchmark. "Meta-Training" refers to the total number of parameters required to meta-train the model, which can differ from the final number of parameters during "Meta-Testing". We additionally offer a component-wise breakdown.

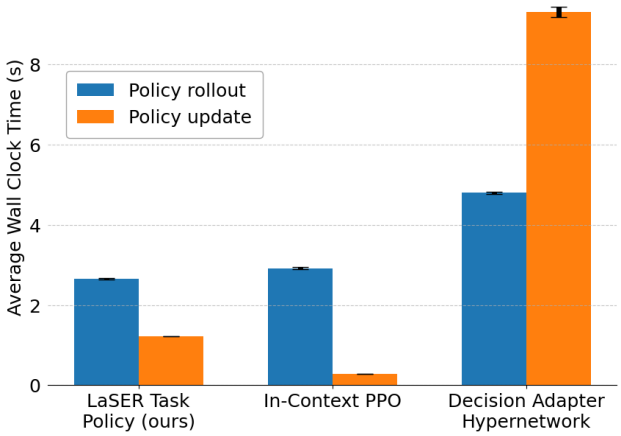

Figure 16: Wall clock time results for task policy meta-training. We measure the average time per iteration (in seconds) required to roll out episodes and optimize the policy. The results are averaged over 10 seeds. The bars represent standard error across all seeds.

General Hyperparameters				
	MEWA	**Meta-World**	**HopperMass**	**Notes**
N_{explore}	10,000	700	600	
$N_{\text{explore}}^{\text{initial}}$	2,000	20	30	
H	50	200	400	Horizon
K	4	2	4	Episodes per meta-episode

Table 5: LaSER hyperparameters used throughout meta-training and meta-testing.

Encoder g_ω				
	MEWA	**Meta-World**	**HopperMass**	**Notes**
Learning rate	1×10^{-4}		5×10^{-4}	
N_{encoder}	50	2		Encoder updates per iteration
Q	20			Meta-episodes per task
$\lvert\mathcal{B}\rvert$	5	10		# of tasks per batch
Buffer size	$100,000$	$1,000$	$2,000$	Maximum # of tasks in buffer
$c_{\text{rec}}, c_{\text{t-rec}}, c_{\text{contr}}, c_{\mathcal{R}}$	0.5, 1, 1, 0.125			Coefficients for loss $\mathcal{L}_{\text{LaSER}}$; Eq. 10
ξ	0.05			Constant in Eq. 7
ν	250	25		Update delay for parameters $\hat{\omega}$

Table 6: LaSER hyperparameters used for meta-training the encoder.

Exploration Policy $\pi_\phi^{\text{explore}}$				
	MEWA	**Meta-World**	**HopperMass**	**Notes**
Learning rate	1×10^{-5}		8×10^{-5}	
σ	0.025	0.01	0.003	Constant in Eq. 4
ϵ	0.1			PPO clipping (exploration); Eq. 13
α_R	0		0.01	Environmental reward weight; Eq. 11

Table 7: LaSER hyperparameters used for meta-training the exploration policy.

Task Policy π_θ				
	MEWA	**Meta-World**	**HopperMass**	**Notes**
Learning rate	1×10^{-4}		6×10^{-5}	
N_{task}	$10,000$	$9,000$	$2,500$	
β	1.5	0		Coefficient in Eq. 2
ζ	-0.13	0		Threshold in Eq. 3
ϵ	0.2			PPO clipping (exploitation); Eq. 13
c_{PFO}	0.1	0		PFO coefficient
N_{PPO}	4	2	4	
# minibatches	2	8		
Layers	$(128, 128, 128)$	$(256, 256, 128)$		
Context encoder layers	$(128, 128, 128)$	$(256, 256, 128, 10)$		Encoder for context vector $\boldsymbol{c}$

Table 8: LaSER hyperparameters used for meta-training the task policy.