Decoupling Exploration and Exploitation in Meta-Reinforcement Learning without Sacrifices

Evan Zheran Liu, Aditi Raghunathan, Percy Liang, Chelsea Finn
Department of Computer Science, Stanford University
{evanliu,aditir,pliang,cbfinn}@cs.stanford.edu

Abstract

The goal of meta-reinforcement learning (meta-RL) is to build agents that can quickly learn new tasks by leveraging prior experience on related tasks. Learning a new task often requires both exploring to gather task-relevant information and exploiting this information to solve the task. In principle, optimal exploration and exploitation can be learned end-to-end by simply maximizing task performance. However, such meta-RL approaches struggle with local optima due to a chicken-and-egg problem: learning to explore requires good exploitation to gauge the exploration's utility, but learning to exploit requires information gathered via exploration. Optimizing separate objectives for exploration and exploitation can avoid this problem, but prior meta-RL exploration objectives yield suboptimal policies that gather information irrelevant to the task. We alleviate both concerns by constructing an exploitation objective that automatically identifies task-relevant information and an exploration objective to recover only this information. This avoids local optima in end-to-end training, without sacrificing optimal exploration. Empirically, DREAM substantially outperforms existing approaches on complex meta-RL problems, such as sparse-reward 3D visual navigation.[1]

1 Introduction

A general-purpose agent should be able to perform multiple related tasks across multiple related environments. Our goal is to develop agents that can perform a variety of tasks in novel environments, based on previous experience and only a small amount of experience in the new environment. For example, we may want a robot to cook a meal (a new task) in a new kitchen (the environment) after it has learned to cook other meals in other kitchens. To adapt to a new kitchen, the robot must both explore to find the ingredients, and use this information to cook. Existing meta-reinforcement learning (meta-RL) methods can adapt to new tasks and environments, but, as we identify in this work, struggle when adaptation requires complex exploration.

In the meta-RL setting, the agent is presented with a set of meta-training problems, each in an environment (e.g., a kitchen) with some task (e.g., make pizza); at meta-test time, the agent is given a new, but related environment and task. It is allowed to gather information in a few initial (exploration) episodes, and its goal is to then maximize returns on all subsequent (exploitation) episodes, using this information. A common meta-RL approach is to learn to explore and exploit *end-to-end* by training a policy and updating exploration behavior based on how well the policy later exploits using the information discovered from exploration [8, 39, 34, 47, 19]. With enough model capacity, such approaches can express optimal exploration and exploitation, but they create a chicken-and-egg problem that leads to bad local optima and poor sample efficiency: Learning to explore requires good exploitation to gauge the exploration's utility, but learning to exploit requires information gathered via exploration; therefore, with only final performance as signal, one cannot be learned without

[1] Project web page: https://anonymouspapersubmission.github.io/dream/

4th Workshop on Meta-Learning at NeurIPS 2020, Vancouver, Canada.

already having learned the other. For example, a robot chef is only incentivized to explore and find the ingredients if it already knows how to cook, but the robot can only learn to cook if it can already find the ingredients by exploration.

To avoid the chicken-and-egg problem, we propose to optimize separate objectives for exploration and exploitation by leveraging the *problem ID*—an easy-to-provide unique one-hot for each training meta-training task and environment. Some prior works [19, 21] also use these problem IDs, but not in a way that avoids the chicken-and-egg problem. Others [28, 46, 13, 14, 44] also optimize separate objectives, but their exploration objectives learn suboptimal policies that gather task-irrelevant information (e.g., the color of the walls). Instead, we propose an exploitation objective that automatically identifies task-relevant information, and an exploration objective to recover only this information. We learn an exploitation policy without the need for exploration, by conditioning on a learned representation of the problem ID, which provides task-relevant information (e.g., by memorizing the locations of the ingredients for each ID / kitchen). We also apply an information bottleneck to this representation to encourage discarding of any information not required by the exploitation policy (i.e., task-irrelevant information). Then, we learn an exploration policy to only discover task-relevant information by training it to produce trajectories containing the same information as the learned ID representation (Section 4). Crucially, unlike prior work, we prove that our separate objectives are *consistent*: optimizing them yields optimal exploration and exploitation, assuming expressive-enough policy classes and enough meta-training data (Section 5.1).

Overall, we present two core contributions: (i) we articulate and formalize a chicken-and-egg coupling problem between optimizing exploration and exploitation in meta-RL (Section 4.1); and (ii) we overcome this with a consistent decoupled approach, called DREAM: **D**ecoupling exploRation and **E**xploitAtion in **M**eta-RL (Section 4.2). Theoretically, in a simple tabular example, we show that addressing the coupling problem with DREAM provably improves sample complexity over existing end-to-end approaches by a factor exponential in the horizon (Section 5). Empirically, we stress test DREAM's ability to learn sophisticated exploration strategies on 3 challenging, didactic benchmarks and a sparse-reward 3D visual navigation benchmark. On these, DREAM learns to optimally explore and exploit, achieving 80% higher returns than existing state-of-the-art approaches (PEARL, E-RL^2, IMPORT, VARIBAD), which struggle to learn an effective exploration strategy (Section 6).

2 Related Work

We draw on a long line of work on learning to adapt to related tasks [33, 36, 27, 5, 4, 16, 2, 32]. Many meta-RL works focus on adapting efficiently to a new task from few samples without optimizing the sample collection process, via updating the policy parameters [11, 1, 42, 18, 23], learning a model [26, 31, 15], multi-task learning [10], or leveraging demonstrations [45]. In contrast, we focus on problems where targeted exploration is critical for few-shot adaptation.

Approaches that specifically explore to obtain the most informative samples fall into two main categories: *end-to-end* and *decoupled* approaches. End-to-end approaches optimize exploration and exploitation end-to-end by updating exploration behavior from returns achieved by exploitation [8, 39, 24, 29, 34, 47, 19, 21, 7]. These approaches can represent the optimal policy [20], but they struggle to escape local optima due to a chicken-and-egg problem between learning to explore and learning to exploit (Section 4.1). Several of these approaches [19, 21] also leverage the problem ID during meta-training, but they still learn end-to-end, so the chicken-and-egg problem remains.

Decoupled approaches instead optimize separate exploration and exploitation objectives, via, e.g., Thompson-sampling (TS) [35, 28], obtaining exploration trajectories predictive of dynamics or rewards [46, 14, 44], or exploration noise [13]. While these works do not identify the chicken-and-egg problem, decoupled approaches coincidentally avoid it. However, existing decoupled approaches, including ones [28] that leverage the problem ID, do not learn optimal exploration: TS [28] explores by guessing the task and executing a policy for that task, and hence cannot represent exploration behaviors that are different from exploitation [30]. Predicting the dynamics [46, 14, 44] is inefficient when only a small subset of the dynamics are relevant to solving the task. In contrast, we propose a separate mutual information objective for exploration, which both avoids the chicken-and-egg problem and yields optimal exploration when optimized (Section 5). Past work [12, 17, 9, 41] also optimize mutual information objectives, but not for meta-RL.

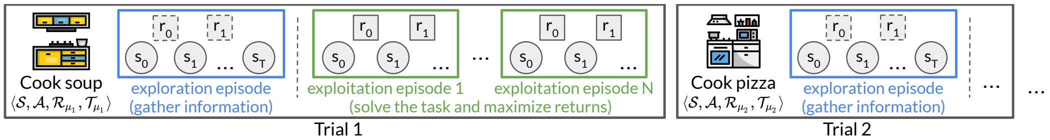

r_0
r_1
s_0
s_1
...
s_T
Cook soup
$\langle \mathcal{S}, \mathcal{A}, \mathcal{R}_{\mu_1}, \mathcal{T}_{\mu_1} \rangle$
exploration episode
(gather information)
exploitation episode 1
exploitation episode N
(solve the task and maximize returns)
Cook pizza
$\langle \mathcal{S}, \mathcal{A}, \mathcal{R}_{\mu_2}, \mathcal{T}_{\mu_2} \rangle$
Trial 1
Trial 2

Figure 1: Meta-RL setting: Given a new environment and task, the agent is allowed to first explore and gather information, and then must use this information to solve the task in subsequent exploitation episodes.

3 Preliminaries

Meta-reinforcement learning. The meta-RL setting considers a family of Markov decision processes (MDPs) $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}_\mu, T_\mu \rangle$ with states $\mathcal{S}$, actions $\mathcal{A}$, rewards $\mathcal{R}_\mu$, and dynamics T_μ, indexed by a one-hot *problem ID* $\mu \in \mathcal{M}$, drawn from a distribution $p(\mu)$. Colloquially, we refer to the dynamics as the *environment*, the rewards as the *task*, and the entire MDP as the *problem*. Borrowing terminology from Duan et al. [8], meta-training and meta-testing both consist of repeatedly running *trials*. Each trial consists of sampling a problem ID $\mu \sim p(\mu)$ and running $N + 1$ episodes on the corresponding problem. Following prior evaluation settings [11, 28, 29, 10], we designate the first episode in a trial as an *exploration* episode consisting of T steps for gathering information, and define the goal as maximizing the returns in the subsequent N *exploitation* episodes (Figure 1). Following Rakelly et al. [28], Humplik et al. [19], Kamienny et al. [21], the easy-to-provide problem ID is available for meta-training, but not meta-testing trials.

We formally express the goal in terms of an exploration policy π^{exp} used in the exploration episode and an exploitation policy π^{task} used in exploitation episodes, but these policies may be the same or share parameters. Rolling out π^{exp} in the exploration episode produces an exploration trajectory $\tau^{\text{exp}} = (s_0, a_0, r_0, \ldots, s_T)$, which contains information discovered via exploration. The exploitation policy π^{task} may then condition on τ^{exp} and optionally, its history across all exploitation episodes in a trial, to maximize exploitation episode returns. The goal is therefore to maximize:

$$\mathcal{J}(\pi^{\text{exp}}, \pi^{\text{task}}) = \mathbb{E}_{\mu \sim p(\mu), \tau^{\text{exp}} \sim \pi^{\text{exp}}} \left[V^{\text{task}}(\tau^{\text{exp}}; \mu) \right], \tag{1}$$

where $V^{\text{task}}(\tau^{\text{exp}}; \mu)$ is the expected returns of π^{task} conditioned on τ^{exp}, summed over the N exploitation episodes in a trial with problem ID μ.

End-to-end meta-RL. A common meta-RL approach [39, 8, 29, 47, 21, 19] is to learn to explore and exploit *end-to-end* by directly optimizing $\mathcal{J}$ in (1), updating both from rewards achieved during exploitation. These approaches typically learn a single recurrent policy $\pi(a_t \mid s_t, \tau_{:t})$ for both exploration and exploitation (i.e., $\pi^{\text{task}} = \pi^{\text{exp}} = \pi$), which takes action a_t given state s_t and history of experiences spanning all episodes in a trial $\tau_{:t} = (s_0, a_0, r_0, \ldots, s_{t-1}, a_{t-1}, r_{t-1})$. Intuitively, this policy is learned by rolling out a trial, producing an exploration trajectory τ^{exp} and, conditioned on τ^{exp} and the exploitation experiences so far, yielding some exploitation episode returns. Then, credit is assigned to both exploration (producing τ^{exp}) and exploitation by backpropagating the exploitation returns through the recurrent policy. Directly optimizing the objective $\mathcal{J}$ this way can learn optimal exploration and exploitation strategies, but optimization is challenging, which we show in Section 4.1.

4 Decoupling Exploration and Exploitation

4.1 The Problem with Coupling Exploration and Exploitation

We begin by showing that end-to-end optimization struggle with local optima due to a chicken-and-egg problem. Figure 2a illustrates this. Learning π^{exp} relies on gradients passed through π^{task}. If π^{task} cannot effectively solve the task, then these gradients will be uninformative. However, to learn to solve the task, π^{task} needs good exploration data (trajectories τ^{exp}) from a good exploration policy π^{exp}. This results in bad local optima as follows: if our current (suboptimal) π^{task} obtains low rewards with a good informative trajectory $\tau^{\text{exp}}_{\text{good}}$, the low reward would cause π^{exp} to learn to *not* generate $\tau^{\text{exp}}_{\text{good}}$. This causes π^{exp} to instead generate trajectories $\tau^{\text{exp}}_{\text{bad}}$ that lack information required to obtain high reward, further preventing the exploitation policy π^{task} from learning. Typically, early in training, both π^{exp} and π^{task} are suboptimal and hence will likely reach this local optimum. In Section 5.2, we illustrate how this local optimum can cause sample inefficiency in a simple example.

4.2 DREAM: Decoupling Exploration and Exploitation in Meta-Learning

While we can sidestep the local optima of end-to-end training by optimizing separate objectives for exploration and exploitation, the challenge is to construct objectives that yield the same optimal

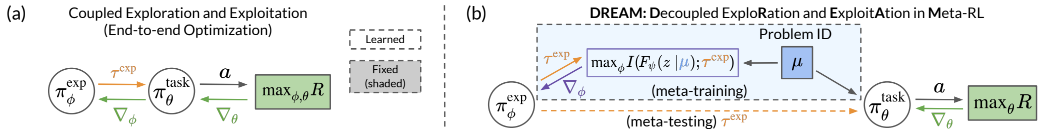

(a)
Coupled Exploration and Exploitation
(End-to-end Optimization)
π_ϕ^{exp}
τ^{exp}
∇_ϕ
π_θ^{task}
a
∇_θ
$\max_{\phi,\theta} R$
Learned
Fixed
(shaded)
(b)
DREAM: Decoupled ExploRation and ExploitAtion in Meta-RL
Problem ID
τ^{exp}
$\max_\phi I(F_\psi(z \mid \mu); \tau^{\text{exp}})$
μ
∇_ϕ
(meta-training)
π_ϕ^{exp}
(meta-testing) τ^{exp}
π_θ^{task}
a
∇_θ
$\max_\theta R$

Figure 2: (a) Coupling between the exploration policy π^{exp} and exploitation policy π^{task}. These policies are illustrated separately for clarity, but may be a single policy. Since the two policies depend on each other (for gradient signal and the τ^{exp} distribution), it is challenging to learn one when the other policy has not learned. (b) DREAM: π^{exp} and π^{task} are learned from decoupled objectives by leveraging a simple one-hot problem ID during meta-training. At meta-test time, the exploitation policy conditions on the exploration trajectory as before.

solution as the end-to-end approach. We now discuss how we can use the easy-to-provide problem IDs during meta-training to do so. A good exploration objective should encourage discovering task-relevant distinguishing attributes of the problem (e.g., ingredient locations), and ignoring task-irrelevant attributes (e.g., wall color). To create this objective, the key idea behind DREAM is to *learn* to extract only the task-relevant information from the problem ID, which encodes all information about the problem. Then, DREAM's exploration objective seeks to recover this task-relevant information.

Concretely, DREAM extracts only the task-relevant information from the problem ID μ via a stochastic encoder $F_\psi(z \mid \mu)$. To learn this encoder, we train an exploitation policy π^{task} to maximize rewards, conditioned on samples z from $F_\psi(z \mid \mu)$, while simultaneously applying an information bottleneck to z to discard information not needed by π^{task} (i.e., task-irrelevant information). Then, DREAM learns an exploration policy π^{exp} to produce trajectories with high mutual information with z. In this approach, the exploitation policy π^{task} no longer relies on effective exploration from π^{exp} to learn, and once $F_\psi(z \mid \mu)$ is learned, the exploration policy also learns independently from π^{task}, decoupling the two optimization processes. During meta-testing, when μ is unavailable, the two policies easily combine, since the trajectories generated by π^{exp} are optimized to contain the same information as the encodings $z \sim F_\psi(z \mid \mu)$ that the exploitation policy π^{task} trained on (overview in Figure 2b).

Learning the problem ID encodings and exploitation policy. We begin with learning a stochastic encoder $F_\psi(z \mid \mu)$ parametrized by ψ and exploitation policy π_θ^{task} parametrized by θ, which conditions on z. We learn F_ψ jointly with π_θ^{task} by optimizing the following objective:

$$\underset{\psi,\theta}{\text{maximize}} \ \underbrace{\mathbb{E}_{\mu \sim p(\mu), z \sim F_\psi(z|\mu)} \left[V^{\pi_\theta^{\text{task}}}(z; \mu) \right]}_{\text{Reward}} - \lambda \underbrace{I(z; \mu)}_{\text{Information bottleneck}}, \tag{2}$$

where $V^{\pi_\theta^{\text{task}}}(z; \mu)$ is the expected return of π_θ^{task} on problem μ given and encoding z. The information bottleneck term encourages discarding any (task-irrelevant) information from z that does not help maximize reward. Importantly, both terms are independent of the exploration policy π^{exp}.

We minimize the mutual information $I(z; \mu)$ by minimizing a variational upper bound on it, $\mathbb{E}_\mu \left[\mathrm{D}_{\mathrm{KL}}(F_\psi(z \mid \mu) || r(z)) \right]$, where r is any prior and z is distributed as $p_\psi(z) = \int_\mu F_\psi(z \mid \mu) p(\mu) d\mu$.

Learning an exploration policy given problem ID encodings. Once we've obtained an encoder $F_\psi(z \mid \mu)$ to extract only the necessary task-relevant information required to optimally solve each task, we can optimize the exploration policy π^{exp} to produce trajectories that contain this same information by maximizing their mutual information $I(\tau^{\text{exp}}; z)$. We slightly abuse notation to use π^{exp} to denote the probability distribution over the trajectories τ^{exp}. Then, the mutual information $I(\tau^{\text{exp}}; z)$ can be efficiently maximized by maximizing a variational lower bound [3] as follows:

$$\begin{aligned} I(\tau^{\text{exp}}; z) &= H(z) - H(z \mid \tau^{\text{exp}}) \geq H(z) + \mathbb{E}_{\mu, z \sim F_\psi, \tau^{\text{exp}} \sim \pi^{\text{exp}}} \left[\log q_\omega(z \mid \tau^{\text{exp}}) \right] \qquad (3) \\ &= H(z) + \mathbb{E}_{\mu, z \sim F_\psi} \left[\log q_\omega(z) \right] + \mathbb{E}_{\mu, z \sim F_\psi, \tau^{\text{exp}} \sim \pi^{\text{exp}}} \left[\sum_{t=1}^{T} \log q_\omega(z \mid \tau_{:t}^{\text{exp}}) - \log q_\omega(z \mid \tau_{:t-1}^{\text{exp}}) \right], \end{aligned}$$

where q_ω is any distribution parametrized by ω. We maximize the above expression over ω to learn q_ω that approximates the true conditional distribution $p(z \mid \tau^{\text{exp}})$, which makes this bound tight. In addition, we do not have access to the problem μ at test time and hence cannot sample from $F_\psi(z \mid \mu)$. Therefore, q_ω serves as a decoder that generates the encoding z from the exploration trajectory τ^{exp}.

Recall, our goal is to maximize (3) w.r.t., trajectories τ^{exp} from the exploration policy π^{exp}. Only the third term depends on τ^{exp}, so we train π^{exp} on rewards set to be this third term (information gain):

$$r_t^{\text{exp}}(a_t, s_{t+1}, \tau_{t-1}^{\text{exp}}; \mu) = \mathbb{E}_{z \sim F_\psi(z|\mu)} \left[\log q_\omega(z \mid [s_{t+1}; a_t; \tau_{:t-1}^{\text{exp}}]) - \log q_\omega(z \mid \tau_{:t-1}^{\text{exp}}) \right] - c. \tag{4}$$

Intuitively, the exploration reward for taking action a_t and transitioning to state s_{t+1} is high if this transition encodes more information about the problem (and hence the encoding $z \sim F_\psi(z \mid \mu)$) than was already present in the trajectory $\tau^{\text{exp}}_{:t-1} = (s_0, a_0, r_0, \dots, s_{t-2}, a_{t-2}, r_{t-2})$. We also include a small penalty c to encourage exploring efficiently in as few timesteps as possible. This reward is attractive because (i) it is independent from the exploitation policy and hence avoids the local optima described in Section 4.1, and (ii) it is dense, so it helps with credit assignment. It is also non-Markov, since it depends on τ^{exp}, so we maximize it with a recurrent $\pi^{\text{exp}}_\phi(a_t \mid s_t, \tau^{\text{exp}}_{:t})$, parametrized by ϕ.

4.3 A Practical Implementation of DREAM

Altogether, DREAM learns four separate neural network components, which we detail below.

1. Encoder $F_\psi(z \mid \mu)$: For simplicity, we parametrize the stochastic encoder by learning a deterministic encoding $f_\psi(\mu)$ and apply Gaussian noise, i.e., $F_\psi(z \mid \mu) = \mathcal{N}(f_\psi(\mu), \rho^2 I)$. We choose a convenient prior $r(z)$ to be a unit Gaussian with same variance $\rho^2 I$, which makes the information bottleneck take the form of simple ℓ_2-regularization $\|f_\psi(\mu)\|_2^2$.
2. Decoder $q_\omega(z \mid \tau^{\text{exp}})$: Similarly, we parametrize the decoder $q_\omega(z \mid \tau^{\text{exp}})$ as a Gaussian centered around a deterministic encoding $g_\omega(\tau^{\text{exp}})$ with variance $\rho^2 I$. Then, q_ω maximizes $\mathbb{E}_{\mu, z \sim F_\psi(z \mid \mu)}\left[\|z - g_\omega(\tau^{\text{exp}})\|_2^2\right]$ w.r.t., ω (Equation 3), and the exploration rewards take the form $r^{\text{exp}}(a, s', \tau^{\text{exp}}; \mu) = \|f_\psi(\mu) - g_\omega([\tau^{\text{exp}}; a; s'])\|_2^2 - \|f_\psi(\mu) - g_\omega([\tau^{\text{exp}}])\|_2^2 - c$ (Equation 4).
3. Exploitation policy π^{task}_θ and 4. Exploration policy π^{exp}_ϕ: We learn both policies with double deep Q-learning [38], treating (s, z) as the state for π^{task}_θ.

For convenience, we jointly learn all components in an EM-like fashion, where in the exploration episode, we assume f_ψ and π^{task}_θ are fixed. There is no chicken-and-egg effect in this joint training because the exploitation policy (along with the stochastic encoder) are trained independent of the exploration policy; only the training of the exploration policy uses the stochastic encodings. Appendix A includes all details and a summary (Algorithm 1).

5 Analysis of DREAM

5.1 Theoretical Consistency of the DREAM Objective

A key property of DREAM is that it is *consistent*: maximizing our decoupled objective also maximizes expected returns (Equation 1). This contrasts prior decoupled approaches [46, 28, 13, 14, 44], which also decouple exploration from exploitation, but do not recover the optimal policy even with infinite data. Formally,

Proposition 1. *Assume $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}_\mu, \mathcal{T}_\mu \rangle$ is ergodic for all problems $\mu \in \mathcal{M}$. Let $V^*(\mu)$ be the maximum expected returns achievable by any exploitation policy with access to the problem ID μ, i.e., with complete information. Let $\pi^{task}_\star, \pi^{exp}_\star, F_\star$ and $q_\star(z \mid \tau^{exp})$ be the optimizers of the* DREAM *objective. Then for long-enough exploration episodes and expressive-enough function classes,*

$$\mathbb{E}_{\mu \sim p(\mu), \tau^{exp} \sim \pi^{exp}_\star, z \sim q_\star(z \mid \tau^{exp})}\left[V^{\pi^{task}_\star}(z; \mu)\right] = \mathbb{E}_{\mu \sim p(\mu)}\left[V^*(\mu)\right].$$

Optimizing DREAM's objective achieves the maximal returns $V^*(\mu)$ even without access to μ during meta-testing (proof in Appendix C.1). We can remove the ergodicity assumption by increasing the number of exploration episodes, and DREAM performs well on non-ergodic MDPs in our experiments.

5.2 An Example Illustrating the Impact of Coupling on Sample Complexity

With enough capacity, end-to-end approaches can also learn the optimal policy, but can be highly sample inefficient due to the coupling problem in Section 4.1. We highlight this in a simple tabular example to remove the effects of function approximation: Each episode is a one-step bandit problem with action space $\mathcal{A}$. Taking action $a_\star$ in the exploration episode leads to a trajectory $\tau^{\text{exp}}_\star$ that reveals the problem ID μ; all other actions a reveal no information and lead to τ^{exp}_a. The ID μ identifies a unique action that receives reward 1 during exploitation; all other actions get reward 0. Therefore, taking $a_\star$ during exploration is necessary and sufficient to obtain optimal reward 1. We now study the number of samples required for RL^2 (the canonical end-to-end approach) and DREAM to learn the optimal exploration policy with ϵ-greedy tabular Q-learning. We precisely describe a more general

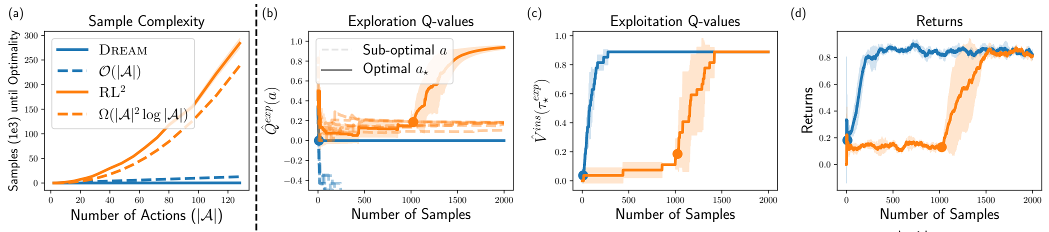


Figure 3: (a) Sample complexity of learning the optimal exploration policy as the action space $|\mathcal{A}|$ grows (1000 seeds). (b) Exploration Q-values $\hat{Q}^{\text{exp}}(a)$. The policy $\arg\max_a \hat{Q}^{\text{exp}}(a)$ is optimal after the dot. (c) Exploitation values given optimal trajectory $\hat{V}^{\text{task}}(\tau_\star^{\text{exp}})$. (d) Returns achieved on a tabular MDP with $|\mathcal{A}| = 8$ (3 seeds).

setup in Appendix C.2 and prove that DREAM *learns the optimal exploration policy in* $\Omega(|\mathcal{A}|^H|\mathcal{M}|)$ *times fewer samples than RL*^{2} in this simple setting with horizon H. Figure 3a empirically validates this result and we provide intuition below.

In the tabular analog of RL^2, the exploitation Q-values form targets for the exploration Q-values: $\hat{Q}^{\text{exp}}(a) \leftarrow \hat{V}^{\text{task}}(\tau_a^{\text{exp}}) := \max_{a'} \hat{Q}^{\text{task}}(\tau_a^{\text{exp}}, a')$. We drop the fixed initial state from notation. This creates the local optimum in Section 4.1. Initially $\hat{V}^{\text{task}}(\tau_\star^{\text{exp}})$ is low, as the exploitation policy has not learned to achieve reward, even when given $\tau_\star^{\text{exp}}$. This causes $\hat{Q}^{\text{exp}}(a_\star)$ to be small and therefore $\arg\max_a \hat{Q}^{\text{exp}}(a) \neq a_\star$ (Figure 3b), which then prevents $\hat{V}^{\text{task}}(\tau_\star^{\text{exp}})$ from learning (Figure 3c) as $\tau_\star^{\text{exp}}$ is roughly sampled only once per $\frac{|\mathcal{A}|}{\epsilon}$ episodes. This effect is mitigated only when $\hat{Q}^{\text{exp}}(a_\star)$ becomes higher than $\hat{Q}^{\text{exp}}(a)$ for the other uninformative a's (the dot in Figure 3b-d). Then, learning both the exploitation and exploration Q-values accelerates, but getting there takes many samples.

In DREAM, the exploration Q-values regress toward the decoder $\hat{q}$: $\hat{Q}^{\text{exp}}(a) \leftarrow \log \hat{q}(\mu \mid \tau^{\text{exp}}(a))$. This decoder learns much faster than $\hat{Q}^{\text{task}}$, since it does not depend on the exploitation actions. Consequently, DREAM's exploration policy quickly becomes optimal (dot in Figure 3b), which enables quickly learning the exploitation Q-values and achieving high reward (Figures 3c and 3d).

6 Experiments

Many real-world problem distributions (e.g., cooking) require exploration (e.g., locating ingredients) that is distinct from exploitation (e.g., cooking these ingredients). Therefore, we desire benchmarks that require distinct exploration and exploitation to stress test aspects of exploration in meta-RL, such as if methods can: (i) efficiently explore, even in the presence of distractions; (ii) leverage informative objects (e.g., a map) to aid exploration; (iii) learn exploration and exploitation strategies that generalize to unseen problems; (iv) scale to challenging exploration problems with high-dimensional visual observations. Existing benchmarks (e.g., MetaWorld [43] or MuJoCo tasks like HalfCheetahVelocity [11, 29]) were not designed to test exploration and are unsuitable for answering these questions. These benchmarks mainly vary the rewards (e.g., the speed to run at) across problems, so naively exploring by exhaustively trying different exploitation behaviors (e.g., running at different speeds) is optimal. They further don't include visual states, distractors, or informative objects, which test if exploration is efficient. We therefore design new benchmarks meeting the above criteria, testing (i-iii) with didactic benchmarks, and (iv) with a sparse-reward 3D visual navigation benchmark, based on Kamienny et al. [21], that combines complex exploration with high-dimensional visual inputs. To further deepen the exploration challenge, we make our benchmarks goal-conditioned. This requires exploring to discover information relevant to *any* potential goal, rather than just a single task (e.g., locating all ingredients for *any* meal vs. just the ingredients for pasta).

Comparisons. We compare DREAM with state-of-the-art end-to-end (E-RL^2 [34], VARIBAD [47], and IMPORT [21]) and decoupled approaches (PEARL-UB, an upper bound on the final performance of PEARL [28]). For PEARL-UB, we analytically compute the expected rewards achieved by optimal Thompson sampling (TS) exploration, assuming access to the optimal problem-specific policy and true posterior problem distribution. Like DREAM, IMPORT and PEARL also use the one-hot problem ID, during meta-training. We also report the optimal returns achievable with no exploration as "No exploration." We report the average returns achieved by each approach in trials with one exploration and one exploitation episode, averaged over 3 seeds with 1-standard deviation error bars (full details in Appendix B).

6.1 Didactic Experiments

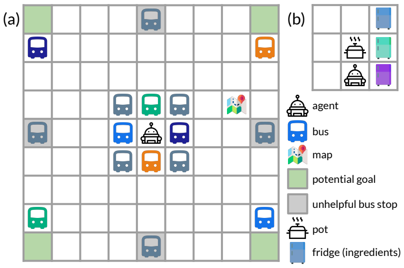


Figure 4: Didactic grid worlds to stress test exploration. (a) Navigation. (b) Cooking.

We first evaluate on the grid worlds shown in Figures 4a and 4b. The state consists of the agent's (x, y)-position, a one-hot indicator of the object at the agent's position (none, bus, map, pot, or fridge), a one-hot indicator of the agent's inventory (none or an ingredient), and the goal. The actions are *move* up, down, left, or right; *ride bus*, which, at a bus, teleports the agent to another bus of the same color; *pick up*, which, at a fridge, fills the agent's inventory with the fridge's ingredients; and *drop*, which, at the pot, empties the agent's inventory into the pot. Episodes consist of 20 timesteps and the agent receives -0.1 reward at each timestep until the goal, described below, is met (details in Appendix B.1; qualitative results in Appendix B.2).

Targeted exploration. We first test if these methods can efficiently explore in the presence of distractions in two versions of the benchmark in Figure 4a: *distracting bus* and *map*. In both, there are 4 possible goals (the 4 green locations). During each episode, a goal is randomly sampled. Reaching the goal yields +1 reward and ends the episode. The 4 colored buses each lead to near a different potential green goal location when ridden and in different problems μ, their destinations are set to be 1 of the $4!$ different permutations. The *distracting bus* version tests if the agent can ignore distractions by including unhelpful gray buses, which are never needed to optimally reach any goal. In different problems, the gray buses lead to different permutations of the gray locations. The *map* version tests if the agent can leverage objects for exploration by including a map that reveals the destinations of the colored buses when touched.

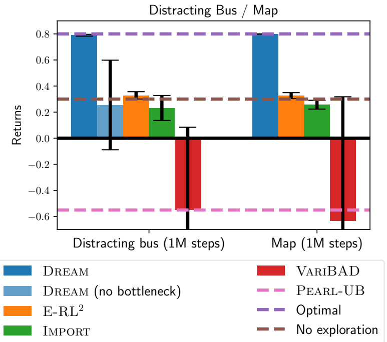


Figure 5: Navigation results. Only DREAM optimally explores all buses and the map.

Figure 5 shows the results after 1M steps. DREAM learns to optimally explore and thus receives optimal reward in both versions: In *distracting bus*, it ignores the unhelpful gray buses and learns the destinations of all helpful buses by riding them. In *map*, it learns to leverage informative objects, by visiting the map and ending the exploration episode. During exploitation, DREAM immediately reaches the goal by riding the correct colored bus. In contrast, IMPORT and E-RL^2 get stuck in a local optimum, indicative of the coupling problem (Section 4.1), which achieves the same returns as no exploration at all. They do not explore the helpful buses or map and consequently sub-optimally exploit by just walking to the goal. VARIBAD learns slower, likely because it learns a dynamics model, but eventually matches the sub-optimal returns of IMPORT and RL^2 in ~3M steps (not shown).

PEARL achieves sub-optimal returns, even with infinite meta-training (see line for PEARL-UB), as follows. TS explores by sampling a problem ID from its posterior and executing its policy conditioned on this ID. Since for any given problem (bus configuration) and goal, the optimal problem-specific policy rides the one bus leading to the goal, TS does not explore optimally (i.e., explore all the buses or read the map), even with the optimal problem-specific policy and true posterior problem distribution.

Recall that DREAM tries to remove extraneous information from the problem ID with an information bottleneck that minimizes the mutual information $I(z; \mu)$ between problem IDs and the encoder $F_\psi(z \mid \mu)$. In *distracting bus*, we test the importance of the information bottleneck by ablating it from DREAM. As seen in Figure 5 (left), this ablation (DREAM (no bottleneck)) wastes its exploration on the gray unhelpful buses, since they are part of the problem, and consequently achieves low returns.

Generalization to new problems. We test generalization to unseen problems in a cooking benchmark (Figure 4b). The fridges on the right each contain 1 of 4 different (color-coded) ingredients, determined by the problem ID. The fridges' contents are unobserved until the agent uses the "pickup" action at the fridge. Goals (recipes) specify placing 2 correct ingredients in the pot in the right order. The agent receives positive reward for picking up and placing the correct ingredients, and negative reward for using the wrong ingredients. We hold out 1 of the $4^3 = 64$ problems from meta-training.

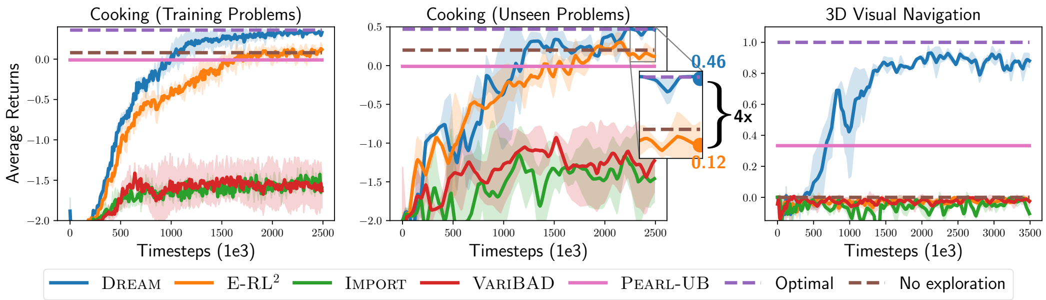


Figure 6: Cooking results. Only DREAM achieves optimal reward on training problems (left), on generalizing to unseen problems (middle). 3D visual navigation results (right). Only DREAM reads the sign and solves the task.

Figure 6 shows the results on training (left) and held-out (middle) problems. Only DREAM achieves near-optimal returns on both. During exploration, it investigates each fridge with the "pick up" action, and then directly retrieves the correct ingredients during exploitation. E-RL^2 gets stuck in a local optimum, only sometimes exploring the fridges. This achieves 3x lower returns, only slightly higher than no exploration at all. Here, leveraging the problem ID actually hurts IMPORT compared to E-RL^2. IMPORT successfully solves the task, given access to the problem ID, but fails without it. As before, VARIBAD learns slowly and TS (PEARL-UB) cannot learn optimal exploration.

6.2 Sparse-Reward 3D Visual Navigation

We conclude with a challenging benchmark testing both sophisticated exploration and scalability to pixel inputs. We modify a benchmark from Kamienny et al. [21] to increase both the exploration and scalability challenge by including more objects and a visual sign, illustrated in Figure 7. In the 3 different problems, the sign on the right says “blue”, “red” or “green.” The goals specify whether the agent should collect the key or block. The agent receives +1 reward for collecting the correct object (color specified by the sign, shape specified by the goal), -1 reward for the wrong object, and 0 reward otherwise. The agent begins the episode on the far side of the barrier and must walk around the barrier to visually “read” the sign. The agent’s observations are 80 × 60 RGB images and its actions are to rotate left or right, move forward, or end the episode.

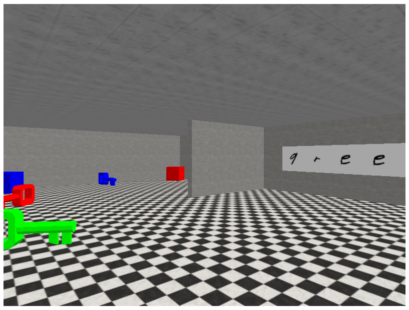


Figure 7: 3D Visual Navigation. The agent must read the sign to determine what colored object to go to.

DREAM is the only method that learns to read the sign and achieve reward (Figure 6 right). All end-to-end approaches get stuck in local optima, where they do not learn to read the sign and hence stay away from all the objects, in fear of receiving negative reward. This achieves close to 0 returns, consistent with the results in Kamienny et al. [21]. As before, PEARL-UB cannot learn optimal exploration.

7 Conclusion

In summary, this work identifies a chicken-and-egg problem that end-to-end meta-RL approaches suffer from, where learning good exploitation requires already having learned good exploration and vice-versa. This creates challenging local optima, since typically neither exploration nor exploitation is good at the beginning of meta-training. We show that appropriately leveraging simple one-hot problem IDs allows us to break this cyclic dependency with DREAM. Consequently, DREAM has strong empirical performance on meta-RL problems requiring complex exploration, as well as substantial theoretical sample complexity improvements in the tabular setting. Though prior works also leverage the problem ID and use decoupled objectives that avoid the chicken-and-egg problem, no other existing approaches can recover optimal exploration empirically and theoretically like DREAM.

Reproducibility. Our code is available at https://github.com/ezliu/dream.

Acknowledgments and Disclosure of Funding

We thank Luisa Zintgraf for her insights about VARIBAD. We are grateful to Suraj Nair, Minae Kwon, and Ramtin Keramati for their feedback on an early draft of this work. We thank Arkira

Chantaratanond, who designed the animated videos of the DREAM agent in the cooking task, available on the project web page. EL is supported by a National Science Foundation Graduate Research Fellowship under Grant No. DGE-1656518. AR is supported by a Google PhD Fellowship and Open Philanthropy Project AI Fellowship.

Icons used in this paper were made by Freepik, ThoseIcons, dDara from www.flaticon.com.

References

[1] R. Agarwal, C. Liang, D. Schuurmans, and M. Norouzi. Learning to generalize from sparse and underspecified rewards. *arXiv preprint arXiv:1902.07198*, 2019.

[2] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. D. Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

[3] D. Barber and F. V. Agakov. The IM algorithm: a variational approach to information maximization. In *Advances in neural information processing systems*, 2003.

[4] S. Bengio, Y. Bengio, J. Cloutier, and J. Gecsei. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, volume 2, 1992.

[5] Y. Bengio, S. Bengio, and J. Cloutier. Learning a synaptic learning rule. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume 2, pages 969–969, 1991.

[6] M. Chevalier-Boisvert. Gym-Miniworld environment for openai gym. https://github.com/maximecb/gym-miniworld, 2018.

[7] R. Dorfman and A. Tamar. Offline meta reinforcement learning. *arXiv preprint arXiv:2008.02598*, 2020.

[8] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[9] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

[10] R. Fakoor, P. Chaudhari, S. Soatto, and A. J. Smola. Meta-Q-learning. *arXiv preprint arXiv:1910.00125*, 2019.

[11] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, 2017.

[12] K. Gregor, D. J. Rezende, and D. Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

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

[14] S. Gurumurthy, S. Kumar, and K. Sycara. Mame: Model-agnostic meta-exploration. *arXiv preprint arXiv:1911.04024*, 2019.

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

[16] S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks (ICANN)*, pages 87–94, 2001.

[17] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1109–1117, 2016.

[18] R. Houthooft, Y. Chen, P. Isola, B. Stadie, F. Wolski, O. J. Ho, and P. Abbeel. Evolved policy gradients. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5400–5409, 2018.

[19] J. Humplik, A. Galashov, L. Hasenclever, P. A. Ortega, Y. W. Teh, and N. Heess. Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*, 2019.

[20] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

[21] P. Kamienny, M. Pirotta, A. Lazaric, T. Lavril, N. Usunier, and L. Denoyer. Learning adaptive exploration strategies in dynamic environments through informed policy regularization. *arXiv preprint arXiv:2005.02934*, 2020.

[22] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2019.

[23] R. Mendonca, A. Gupta, R. Kralev, P. Abbeel, S. Levine, and C. Finn. Guided meta-policy search. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 9653–9664, 2019.

[24] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*, 2017.

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

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

[27] D. K. Naik and R. J. Mammone. Meta-neural networks that learn by learning. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 1, pages 437–442, 1992.

[28] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. *arXiv preprint arXiv:1903.08254*, 2019.

[29] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel. Promp: Proximal meta-policy search. *arXiv preprint arXiv:1810.06784*, 2018.

[30] D. Russo, B. V. Roy, A. Kazerouni, I. Osband, and Z. Wen. A tutorial on thompson sampling. *arXiv preprint arXiv:1707.02038*, 2017.

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

[32] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

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

[34] B. Stadie, G. Yang, R. Houthooft, P. Chen, Y. Duan, Y. Wu, P. Abbeel, and I. Sutskever. The importance of sampling inmeta-reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 9280–9290, 2018.

[35] W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3):285–294, 1933.

[36] S. Thrun and L. Pratt. *Learning to learn*. Springer Science & Business Media Springer Science & Business Media, 2012.

[37] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(0):2579–2605, 2008.

[38] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-learning. In *Association for the Advancement of Artificial Intelligence (AAAI)*, volume 16, pages 2094–2100, 2016.

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

[40] Z. Wang, T. Schaul, M. Hessel, H. V. Hasselt, M. Lanctot, and N. D. Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2016.

[41] D. Warde-Farley, T. V. de Wiele, T. Kulkarni, C. Ionescu, S. Hansen, and V. Mnih. Unsupervised control through non-parametric discriminative rewards. *arXiv preprint arXiv:1811.11359*, 2018.

[42] Y. Yang, K. Caluwaerts, A. Iscen, J. Tan, and C. Finn. Norml: No-reward meta learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 323–331, 2019.

[43] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. *arXiv preprint arXiv:1910.10897*, 2019.

[44] J. Zhang, J. Wang, H. Hu, Y. Chen, C. Fan, and C. Zhang. Learn to effectively explore in context-based meta-RL. *arXiv preprint arXiv:2006.08170*, 2020.

[45] A. Zhou, E. Jang, D. Kappler, A. Herzog, M. Khansari, P. Wohlhart, Y. Bai, M. Kalakrishnan, S. Levine, and C. Finn. Watch, try, learn: Meta-learning from demonstrations and reward. *arXiv preprint arXiv:1906.03352*, 2019.

[46] W. Zhou, L. Pinto, and A. Gupta. Environment probing interaction policies. *arXiv preprint arXiv:1907.11740*, 2019.

[47] L. Zintgraf, K. Shiarlis, M. Igl, S. Schulze, Y. Gal, K. Hofmann, and S. Whiteson. Varibad: A very good method for bayes-adaptive deep RL via meta-learning. *arXiv preprint arXiv:1910.08348*, 2019.

A DREAM Training Details

Algorithm 1 summarizes a practical algorithm for training DREAM. Unlike end-to-end approaches, we choose not to make π_θ^{task} recurrent for simplicity, and only condition on z and the current state s. We parametrize the policies as deep dueling double-Q networks [40, 38], with exploration Q-values $\hat{Q}^{\text{exp}}(s, \tau^{\text{exp}}, a; \phi)$ parametrized by ϕ (and target network parameters ϕ') and exploitation Q-values $\hat{Q}^{\text{task}}(s, z, a; \theta)$ parametrized by θ (and target network parameters θ'). We train on trials with one exploration and one exploitation episode, but can test on arbitrarily many exploitation episodes, as the exploitation policy acts on each episode independently (i.e. it does not maintain a hidden state across episodes). Using the choices for F_ψ and q_ω in Section 4.3, training proceeds as follows.

We first sample a new problem for the trial and roll-out the exploration policy, adding the roll-out to a replay buffer (lines 7-9). Then, we roll-out the exploitation policy, adding the roll-out to a separate replay buffer (lines 10-12). We train the exploitation policy on both stochastic encodings of the problem ID $\mathcal{N}(f_\psi(\mu), \rho^2 I)$ and on encodings of the exploration trajectory $g_\omega(\tau^{\text{exp}})$.

Next, we sample from the replay buffers and update the parameters. First, we sample $(s_t, a_t, s_{t+1}, \mu, \tau^{\text{exp}})$-tuples from the exploration replay buffer and perform a normal DDQN update on the exploration Q-value parameters ϕ using rewards computed from the decoder (lines 13-15). Concretely, we minimize the following standard DDQN loss function w.r.t., the parameters ϕ, where the rewards are computed according to Equation 4:

$$\mathcal{L}_{\text{exp}}(\phi) = \mathbb{E}\left[\left\|\hat{Q}^{\text{exp}}(s_t, \tau^{\text{exp}}_{:t-1}, a_t; \phi) - (r_t^{\text{exp}} + \gamma \hat{Q}^{\text{exp}}(s_{t+1}, [\tau^{\text{exp}}_{:t-1}; a_t; s_t], a_{\text{DDQN}}; \phi')\right\|_2^2\right],$$

$$\text{where } r_t^{\text{exp}} = \left\|f_\psi(\mu) - g_\omega(\tau^{\text{exp}}_{:t})\right\|_2^2 - \left\|f_\psi(\mu) - g_\omega(\tau^{\text{exp}}_{:t-1})\right\|_2^2 - c$$

$$\text{and } a_{\text{DDQN}} = \arg\max_a \hat{Q}^{\text{exp}}(s_{t+1}, [\tau^{\text{exp}}_{:t-1}; a_t; s_t]; \phi).$$

We perform a similar update with the exploitation Q-value parameters (lines 16-18). We sample $(s, a, r, s', \mu, \tau^{\text{exp}})$-tuples from the exploitation replay buffer and perform two DDQN updates, one from the encodings of the exploration trajectory and one from the encodings of the problem ID by minimizing the following losses:

$$\mathcal{L}_{\text{task-traj}}(\theta, \omega) = \mathbb{E}\left[\left\|\hat{Q}^{\text{task}}(s, g_\omega(\tau^{\text{exp}}), a; \theta) - (r + \hat{Q}^{\text{task}}(s', g_{\omega'}(\tau^{\text{exp}}), a_{\text{traj}}; \theta')\right\|_2^2\right],$$

$$\text{and } \mathcal{L}_{\text{task-id}}(\theta, \psi) = \mathbb{E}\left[\left\|\hat{Q}^{\text{task}}(s, f_\psi(\mu), a; \theta) - (r + \hat{Q}^{\text{task}}(s', f_{\psi'}(\mu), a_{\text{prob}}; \theta')\right\|_2^2\right],$$

$$\text{where } a_{\text{traj}} = \arg\max_a \hat{Q}^{\text{task}}(s', g_\omega(\tau^{\text{exp}}), a; \theta) \text{ and } a_{\text{prob}} = \arg\max_a \hat{Q}^{\text{task}}(s', f_\psi(\mu), a; \theta).$$

Finally, from the same exploitation replay buffer samples, we also update the problem ID embedder to enforce the information bottleneck (line 19) and the decoder to approximate the true conditional distribution (line 20) by minimizing the following losses respectively:

$$\mathcal{L}_{\text{bottleneck}}(\psi) = \mathbb{E}_\mu\left[\min\left(\|f_\psi(\mu)\|_2^2, K\right)\right]$$

$$\text{and } \mathcal{L}_{\text{decoder}}(\omega) = \mathbb{E}_{\tau^{\text{exp}}}\left[\sum_t \left\|f_\psi(\mu) - g_\omega(\tau^{\text{exp}}_{:t})\right\|_2^2\right].$$

Since the magnitude $\|f_\psi(\mu)\|_2^2$ partially determines the scale of the reward, we add a hyperparameter K and only minimize the magnitude when it is larger than K. Altogether, we minimize the following loss:

$$\mathcal{L}(\phi, \theta, \omega, \psi) = \mathcal{L}_{\text{exp}}(\phi) + \mathcal{L}_{\text{task-traj}}(\theta, \omega) + \mathcal{L}_{\text{task-id}}(\theta, \psi) + \mathcal{L}_{\text{bottleneck}}(\psi) + \mathcal{L}_{\text{decoder}}(\omega).$$

As is standard with deep Q-learning [25], instead of sampling from the replay buffers and updating after each episode, we sample and perform all of these updates every 4 timesteps. We periodically update the target networks (lines 21-22).

Algorithm 1 DREAM DDQN

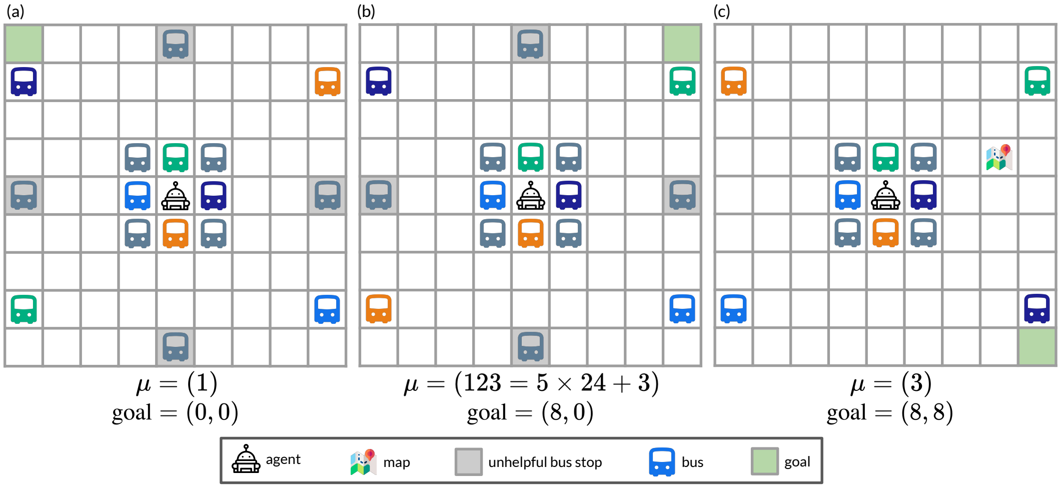
```
**Initialize** exploitation replay buffer $\mathcal{B}_{\text{task-id}} = \{\}$ and exploration replay buffer $\mathcal{B}_{\text{exp}} = \{\}$
**Initialize** exploitation Q-value $\hat{Q}^{\text{task}}$ parameters $\theta$ and target network parameters $\theta'$
**Initialize** exploration Q-value $\hat{Q}^{\text{exp}}$ parameters $\phi$ and target network parameters $\phi'$
**Initialize** problem ID embedder $f_\psi$ parameters $\psi$ and target parameters $\psi'$
**Initialize** trajectory embedder $g_\omega$ parameters $\omega$ and target parameters $\omega'$
**for** trial $= 1$ **to** max trials **do**
    Sample problem $\mu \sim p(\mu)$, defining MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}_\mu, T_\mu \rangle$
    Roll-out $\epsilon$-greedy exploration policy $\hat{Q}^{\text{exp}}(s_t, \tau_{:t}^{\text{exp}}, a_t; \phi)$, producing trajectory $\tau^{\text{exp}} = (s_0, a_0, \dots, s_T)$.
    Add tuples to the exploration replay buffer $\mathcal{B}_{\text{exp}} = \mathcal{B}_{\text{exp}} \cup \{(s_t, a_t, s_{t+1}, \mu, \tau^{\text{exp}})\}_t$.

    Randomly select between embedding $z \sim \mathcal{N}(f_\psi(\mu), \rho^2 I)$ and $z = g_\omega(\tau^{\text{exp}})$.
    Roll-out $\epsilon$-greedy exploitation policy $\hat{Q}^{\text{task}}(s_t, z, a_t; \theta)$, producing trajectory $(s_0, a_0, r_0, \dots)$ with
        $r_t = \mathcal{R}_\mu(s_{t+1})$.
    Add tuples to the exploitation replay buffer $\mathcal{B}_{\text{task-id}} = \mathcal{B}_{\text{task-id}} \cup \{(s_t, a_t, r_t, s_{t+1}, \mu, \tau^{\text{exp}})\}_t$.

    Sample batches of $(s_t, a_t, s_{t+1}, \mu, \tau^{\text{exp}}) \sim \mathcal{B}_{\text{exp}}$ from exploration replay buffer.
    Compute reward $r_t^{\text{exp}} = \|f_\psi(\mu) - g_\omega(\tau_{:t}^{\text{exp}})\|_2^2 - \left\|f_\psi(\mu) - g_\omega(\tau_{:t-1}^{\text{exp}})\right\|_2^2 - c$ (Equation 4).
    Optimize $\phi$ with DDQN update with tuple $(s_t, a_t, r_t^{\text{exp}}, s_{t+1})$ with $\mathcal{L}_{\text{exp}}(\phi)$

    Sample batches of $(s, a, r, s', \mu, \tau^{\text{exp}}) \sim \mathcal{B}_{\text{task-id}}$ from exploitation replay buffer.
    Optimize $\theta$ and $\omega$ with DDQN update with tuple $((s, \tau^{\text{exp}}), a, r, (s', \tau^{\text{exp}}))$ with $\mathcal{L}_{\text{task-traj}}(\theta, \omega)$
    Optimize $\theta$ and $\psi$ with DDQN update with tuple $((s, \mu), a, r, (s', \mu))$ with $\mathcal{L}_{\text{task-id}}(\theta, \psi)$
    Optimize $\psi$ on $\mathcal{L}_{\text{bottleneck}}(\psi) = \nabla_\psi \min(\|f_\psi(\mu)\|_2^2, K)$
    Optimize $\omega$ on $\mathcal{L}_{\text{decoder}}(\omega) = \nabla_\omega \sum_t \|f_\psi(\mu) - g_\omega(\tau_{:t}^{\text{exp}})\|_2^2$ (Equation 3)

    **if** trial $\equiv 0 \pmod{\text{target freq}}$ **then**
        Update target parameters $\phi' = \phi, \theta' = \theta, \psi' = \psi, \omega' = \omega$
    **end if**
**end for**
```



$\mu = (1)$
goal $= (0, 0)$

$\mu = (123 = 5 \times 24 + 3)$
goal $= (8, 0)$

$\mu = (3)$
goal $= (8, 8)$

Figure 8: Examples of different *distracting bus* and *map* problems. (a) An example distracting bus problem. Though all unhelpful distracting buses are drawn in the same color (gray), the destinations of the gray buses are fixed within a problem. (b) Another example distracting bus problem. The destinations of the helpful colored buses are a different permutation (the orange and green buses have swapped locations). This takes on permutation $3 \equiv \mu \pmod{4!}$, instead of 1. The unhelpful gray buses are also a different permutation (not shown), taking on permutation $5 = \left\lfloor \frac{\mu}{4!} \right\rfloor$. (c) An example map problem. Touching the map reveals the destinations of the colored buses, by adding μ to the state observation.

B Experiment Details

B.1 Problem Details

Distracting bus / map. Riding each of the four colored buses teleports the agent to near one of the green goal locations in the corners. In different problems, the destinations of the colored buses change, but the bus positions and their destinations are fixed within each problem. Additionally, in the distracting bus domain, the problem ID also encodes the destinations of the gray buses, which are permutations of the four gray locations on the midpoints of the sides. More precisely, the problem ID $\mu \in \{0, 1, \ldots, 4! \times 4! = 576\}$ encodes both the permutation of the colored helpful bus destinations, indexed as $\mu \pmod{4!}$ and the permutation of the gray unhelpful bus destinations as $\lfloor \frac{\mu}{4!} \rfloor$. We hold out most of the problem IDs during meta-training ($\frac{23}{24} \times 576 = 552$ are held-out for meta-training).

In the map domain, the problem μ is an integer representing which of the $4!$ permutations of the four green goal locations the colored buses map to. The states include an extra dimension, which is set to 0 when the agent is not at the map, and is set to this integer value μ when the agent is at the map. Figure 8 displays three such examples.

Cooking. In different problems, the (color-coded) fridges contain 1 of 4 different ingredients. The ingredients in each fridge are unknown until the goes to the fridge and uses the pickup action. Figure 9 displays three example problems. The problem ID μ is an integer between 0 and 4^3, where $\mu = 4^2a + 4b + c$ indicates that the top right fridge has ingredient a, the middle fridge has ingredient b and the bottom right fridge has ingredient c.

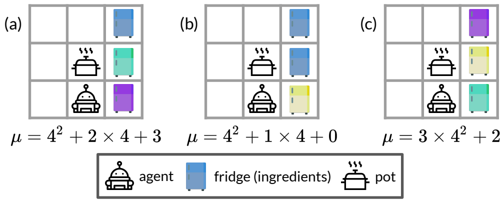


Figure 9: Three example cooking problems. The contents of the fridges (color-coded) are different in different problems.

The goals correspond to a recipe of placing the two correct ingredients in the pot in the right order. Goals are tuples (a, b), which indicate placing ingredient a in the pot first, followed by ingredient b. In a given problem, we only sample goals involving the recipes actually present in that problem. During meta-training, we hold out a randomly selected problem $\mu = 11$.

We use the following reward function $\mathcal{R}_\mu$. The agent receives a per timestep penalty of -0.1 reward and receives $+0.25$ reward for completing each of the four steps: (i) picking up the first ingredient specified by the goal; (ii) placing the first ingredient in the pot; (iii) picking up the second ingredient specified by the goal; and (iv) placing the second ingredient in the pot. To prevent the agent from gaming the reward function, e.g., by repeatedly picking up the first ingredient, dropping the first ingredient anywhere but in the pot yields a penalty of -0.25 reward, and similarly for all steps. To encourage efficient exploration, the agent also receives a penalty of -0.25 reward for picking up the wrong ingredient.

Cooking without goals. While we evaluate on goal-conditioned benchmarks to deepen the exploration challenge, forcing the agent to discover all the relevant information for *any* potential goal, many standard benchmarks [11, 43] don't involve goals. We therefore include a variant of the cooking task, where there are no goals. We simply concatenate the goal (recipe) to the problem ID μ. Additionally, we modify the rewards so that picking up the second ingredient yields $+0.25$ and dropping it yields -0.25 reward, so that it is possible to infer the recipe from the rewards. Finally, to make the problem harder, the agent cannot pick up new ingredients unless its inventory is empty (by using the drop action), and we also increase the number of ingredients to 7. The results are in Section B.2.

Sparse-reward 3D visual navigation. We implement this domain in Gym MiniWorld [6], where the agent's observations are $80 \times 60 \times 3$ RGB arrays. There are three problems $\mu = 0$ (the sign says "blue"), $\mu = 1$ (the sign says "red"), and $\mu = 2$ (the sign says "green"). There are two goals, represented as 0 and 1, corresponding to picking up the key and the box, respectively. The reward function $\mathcal{A}_\mu(s, i)$ is +1 for picking up the correct colored object (according to μ) and the correct type of object (according to the goal) and -1 for picking up an object of the incorrect color or type. Otherwise, the reward is 0. On each episode, the agent begins at a random location on the other side of the barrier from the sign.

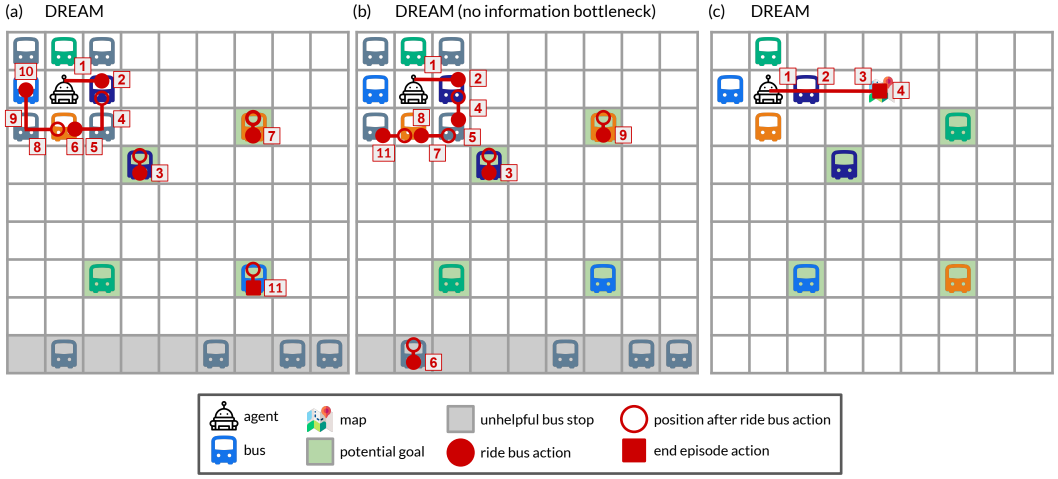


Figure 10: Examples of DREAM's learned exploration behavior. (a) DREAM learns the optimal exploration behavior on the *distraction* variant: riding 3 of the 4 helpful colored buses, which allows it to infer the destinations of all colored buses and efficiently reach any goal during exploitation episodes. (a) Without the information bottleneck, DREAM also explores the unhelpful gray buses, since they are part of the problem. This wastes exploration steps, and leads to lower returns during exploitation episodes. (c) DREAM learns the optimal exploration on the *map* variant: it goes to read the map revealing all the buses' destinations, and then ends the episode.

B.2 Additional Results

Distracting bus / map. Figure 10 shows the exploration policy DREAM learns on the distracting bus and map domains. With the information bottleneck, DREAM optimally explores by riding 3 of the 4 colored buses and inferring the destination of the last colored bus (Figure 8a). Without the information bottleneck, DREAM explores the unhelpful gray buses and runs out of time to explore all of the colored buses, leading to lower reward (Figure 8b). In the map domain, DREAM optimally explores by visiting the map and terminating the exploration episode. In contrast, the other methods (RL^2, IMPORT, VARIBAD) rarely visit the colored buses or map during exploration and consequently walk to their destination during exploitation, which requires more timesteps and therefore receives lower returns.

In Figure 11, we additionally visualize the exploration trajectory encodings $g_\omega(\tau^{\text{exp}})$ and problem ID encodings $f_\psi(\mu)$ that DREAM learns in the distracting bus domain by applying t-SNE [37]. We visualize the encodings of all possible problem IDs as dots. They naturally cluster into $4! = 24$ clusters, where the problems within each cluster differ only in the destinations of the gray distracting buses, and not the colored buses. Problems in the support of the true posterior $p(\mu \mid \tau^{\text{exp}})$ are drawn in green, while problems outside the support (e.g., a problem that specifies that riding the green bus goes to location $(0, 1)$ when it has already been observed in τ^{exp} that riding the orange bus goes to location $(0, 1)$) are drawn in red. We also plot the encoding of the exploration trajectory τ^{exp} so far as a blue cross and the mean of the green clusters as a black square. We find that the encoding of the exploration trajectory $g_\omega(\tau^{\text{exp}})$ tracks the mean of the green clusters until the end of the exploration episode, when only one cluster remains, and the destinations of all the colored buses has been discovered. Intuitively, this captures uncertainty in what the potential problem ID may be. More precisely, when the decoder is a Gaussian, placing $g_\omega(\tau^{\text{exp}})$ at the center of the encodings of problems in the support exactly minimizes Equation 3.

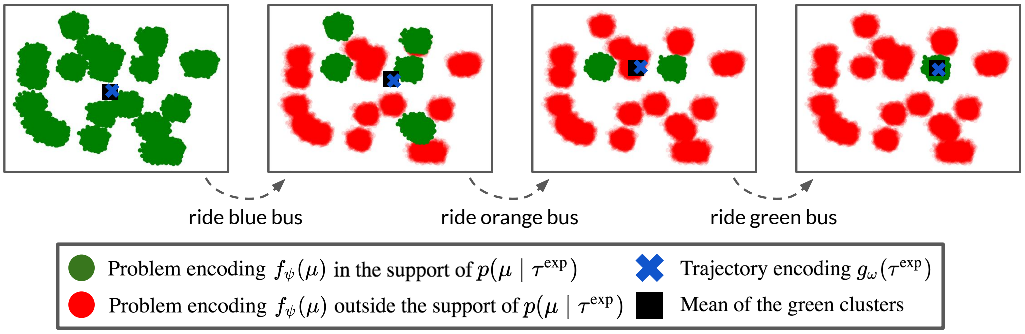


Figure 11: DREAM's learned encodings of the exploration trajectory and problems visualized with t-SNE [37].

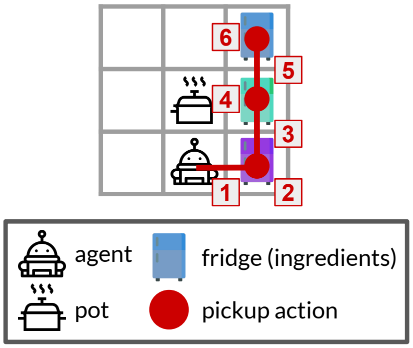


Figure 12: DREAM learns the optimal exploration policy, which learns the fridges' contents by going to each fridge and using the pickup action.

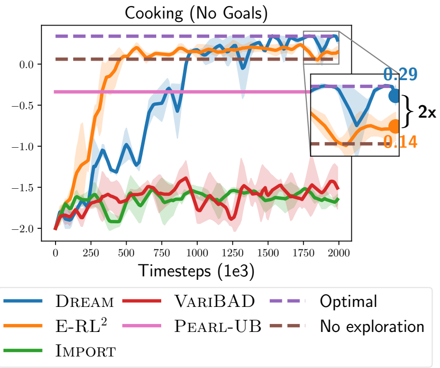


Figure 13: Cooking without goals results. Only DREAM learns the optimal policy, achieving ~2x more reward than the next best approach.

Cooking. Figure 12 shows the exploration policy DREAM learns on the cooking domain, which visits each of the fridges and investigates the contents with the "pickup" action. In contrast, the other methods rarely visit the fridges during exploration, and instead determine the locations of the ingredients during exploitation, which requires more timesteps and therefore receives lower returns.

Cooking without goals. We provide additional results in the case where the cooking domain is modified to not include goals (see Section B.1). The results are summarized in Figure 13 and show the same trends as the results in original cooking benchmark. DREAM learns to optimally explore by investigating the fridges, and then also optimally exploits, by directly collecting the relevant ingredients. The next best approach E-RL^2, only sometimes explores the fridges, again getting stuck in a local optimum, yielding only slightly higher reward than no exploration at all.

Sparse-reward 3D visual navigation. DREAM optimally explores by walking around the barrier and reading the sign. See `https://anonymouspapersubmission.github.io/dream/` for videos. The other methods do not read the sign at all and therefore cannot solve the problem.

B.3 Other Approaches and Architecture Details

In this section, we detail the loss functions that E-RL^2, IMPORT, and VARIBAD optimize, as well as the model architectures used in our experiments. Where possible, we use the same model architecture for all methods: DREAM, E-RL^2, IMPORT, and VARIBAD.

State and problem ID embeddings. All approaches use the same method to embed the state and problem ID. For these embeddings, we embed each dimension independently with an embedding matrix of output dimension 32. Then, we concatenate the per-dimension embeddings and apply two linear layers with output dimensions 256 and 64 respectively, with ReLU activations.

In the 3D visual navigation task, we use a different embedding scheme for the states, as they are images. We apply 3 CNN layers, each with 32 output layers and stride length 2, and with kernel sizes of 5, 5, and 4 respectively. We apply ReLU activations between the CNN layers and apply a final linear layer to the flattened output of the CNN layers, with an output dimension of 128.

All state and problem ID embeddings below use this scheme.

Experience embeddings. E-RL^2, IMPORT, VARIBAD and the exploration policy in DREAM also learn an embedding of the history of prior experiences $\tau^{\text{exp}} = (s_0, a_0, r_0, s_1, \ldots)$ and current state s_T. To do this, we first separately embed each (s_{t+1}, a_t, r_t, d_t)-tuple, where d_t is an episode termination flag (true if the episode ends on this experience, and false otherwise), as follows:

- Embed the state s_t as $e(s_t)$, using the state embedding scheme described above.
- Embed the action a_t as $e(a_t)$ with an embedding matrix of output dimension 16. We set a_{-1} to be 0.
- For the standard meta-RL setting and during exploitation episodes, embed the rewards with a linear layer of output dimension 16. With reward-free exploration in IMRL, the rewards are not embedded in the exploration policy of DREAM, and are embedded as 0 reward for the other approaches, since the same policy is used during both exploration and exploitation. We set r_{-1} to be 0.
- Embed the episode termination d_t as $e(d_t)$ with an embedding matrix of output dimension 16. Note that d is true during all episode terminations within a trial for RL^2, IMPORT, and VARIBAD.

Then, we apply a final linear layer with output dimension 64 to the concatenation of the above $[e(s_t); e(a_t); e(r_t); d_t]$. Finally, to obtain an embedding of the entire history τ^{exp}, we embed each experience separately as above, and then pass an LSTM with hidden dimension 64 over the experience embeddings, where the initial hidden and cell states are set to be 0-vectors.

DREAM. For the decoder $g_\omega(\tau^{\text{exp}} = (s_0, a_0, r_0, s_1, \ldots, s_T))$, we embed each transition (s_t, a_t, r_t, s_{t+1}) of the exploration trajectory τ^{exp} using the same embedding scheme as above, except we also embed the next state s_{t+1}. We do not embed the rewards in the reward-free adaptation version of IMRL. Then, given embeddings for each transition, we embed the entire trajectory by passing an LSTM with output dimension 128 on top of the transition embeddings, followed by two linear layers of output dimension 128 and 64 with ReLU activations.

For the exploitation policy Q-values $\hat{Q}_\theta^{\text{task}}(a \mid s, z)$, we either choose z to be the decoder embedding of the exploration trajectory $g_\omega(\tau^{\text{exp}})$ or to be an embedding of the problem ID $e_\theta(\mu)$, where we always use the exploration trajectory embedding $g_\omega(\tau^{\text{exp}})$ at meta-test time. We embed the state with a learned embedding functions $e(s)$. Then we apply a linear layer of output dimension 64 to the concatenation of $[e(s); z]$ with a ReLU activation. Finally, we apply two linear layer heads of output dimension 1 and $|\mathcal{A}|$ respectively to form estimates of the value and advantage functions, using the dueling Q-network parametrization. To obtain Q-values, we add the value function to the advantage function, subtracting the mean of the advantages.

For the exploration policy Q-values $\hat{Q}_\phi^{\text{exp}}(a_t \mid s_t, \tau_{:t}^{\text{exp}})$, we embed the s_t and $\tau_{:t}^{\text{exp}}$ according to the embedding scheme above. Then, we apply two linear layer heads to obtain value and advantage estimates as above.

E-RL^2. E-RL^2 learns a policy $\pi(a_t \mid s_t, \tau_{:t})$ producing actions a_t given the state s_t and history $\tau_{:t}$. Like with all approaches, we parametrize this with dueling double Q-networks, learning Q-values $\hat{Q}(s_t, \tau_{:t}, a_t)$. We embed the current state s_t and history $\tau_{:t}$ using the embedding scheme described above (with episode termination embeddings). Then, we apply two final linear layer heads to obtain value and advantage estimates.

IMPORT IMPORT also learns a recurrent policy $\pi(a_t \mid s_t, z)$, but conditions on the embedding z, which is either an embedding of the problem μ or the history $\tau_{:t}$. We also parametrize this policy with dueling double Q-networks, learning Q-values $\hat{Q}(s_t, z, a_t)$. We embed the state s_t as $e(s_t)$, the problem μ as $e_\phi(\mu)$ and the history $\tau_{:t}$ as $e_\theta(\tau_{:t})$ using the previously described embedding schemes.

Hyperparameter	Value
Discount Factor γ	0.99
Test-time ϵ	0
Learning Rate	0.0001
Replay buffer batch size	32
Target parameters syncing frequency	5000 updates
Update frequency	4 steps
Grad norm clipping	10

Table 1: Hyperparameters shared across all methods: DREAM, RL^2, IMPORT, and VARIBAD.

Then we alternate meta-training trials between choosing $z = e_\phi(\mu)$ and $z = e_\theta(\tau_{:t})$. We apply a linear layer of output dimension 64 to the concatenation $[e(s_t); z]$ with ReLU activations and then apply two linear layer heads to obtain value and advantage estimates.

Additionally, IMPORT uses the following auxiliary loss function to encourage the history embedding $e_\theta(\tau_{:t})$ to be close to the problem embedding $e_\phi(\mu)$ (optimized only w.r.t., θ):

$$\mathcal{L}_{\text{IMPORT}}(\theta) = \beta \mathbb{E}_{(\tau,\mu)} \left[\sum_t \| e_\theta(\tau_{:t}) - e_\phi(\mu) \|_2^2 \right],$$

where τ is a trajectory from rolling out the policy on problem μ. Following Kamienny et al. [21], we use $\beta = 1$ in our final experiments, and found that performance changed very little when we experimented with other values of β.

VARIBAD. VARIBAD also learns a recurrent policy $\pi(a_t \mid z)$, but over a *belief state* z capturing the history $\tau_{:t}$ and current state s_t. We also parametrize this dueling double Q-networks, learning Q-values $\hat{Q}(s_t, z, a_t)$.

VARIBAD encodes the belief state with an encoder enc$(z \mid s_t, \tau\colon t)$. Our implementation of this encoder embeds s_t and $\tau_{:t}$ using the same experience embedding approach as above, and use the output as the mean m for a distribution. Then, we set enc$(z \mid s_t, \tau\colon t) = \mathcal{N}(m, \nu^2 I)$, where $\nu^2 = 0.00001$. We also tried learning the variance instead of fixing it to $\nu^2 I$ by applying a linear head to the output of the experience embeddings, but found no change in performance, so stuck with the simpler fixed variance approach. Finally, after sampling z from the encoder, we also embed the current state s_t as $e(s_t)$ and apply a linear layer of output dimension 64 to the concatenation $[e(s_t); z]$. Then, we apply two linear layer heads to obtain value and advantage estimates.

VARIBAD does not update the encoder via gradients through the policy. Instead, VARIBAD jointly trains the encoder with state decoder $\hat{T}(s' \mid a, s, z)$ and reward decoder $\hat{\mathcal{R}}(s' \mid a, s, z)$, where z is sampled from the encoder, as follows. Both decoders embed the action a as $e(a)$ with an embedding matrix of output dimension 32 and embed the state s as $e(s)$. Then we apply two linear layers with output dimension 128 to the concatenation $[e(s); e(a); z]$. Finally, we apply two linear heads, one for the state decoder and one for the reward decoder and take the mean-squared error with the true next state s' and the true rewards r respectively. In the 3D visual navigation domain, we remove the state decoder, because the state is too high-dimensional to predict. Note that Zintgraf et al. [47] found better results when removing the state decoder in all experiments. We also tried to remove the state decoder in the grid world experiments, but found better performance when keeping the state decoder. We also found that VARIBAD performed better without the KL loss term, so we excluded that for our final experiments.

B.4 Hyperparameters

In this section, we detail the hyperparameters used in our experiments. Where possible, we used the default DQN hyperparameter values from Mnih et al. [25]. and shared the same hyperparameter values across all methods for fairness. We optimize all methods with the Adam optimizer [**?**]. Table 1 summarizes the shared hyperparameters used by all methods and we detail any differences in hyperparameters between the methods below.

All methods use a linear decaying ϵ schedule for ϵ-greedy exploration. For RL^2, IMPORT, and VARIBAD, we decay ϵ from 1 to 0.01 over 500000 steps. For DREAM, we split the decaying between the exploration and exploitation policies. We decay each policy's ϵ from 1 to 0.01 over 250000 steps.

We train the recurrent policies (DREAM's exploration policy, RL^2, IMPORT, and VARIBAD) with a simplified version of the methods in Kapturowski et al. [22] by storing a replay buffer with up to 16000 sequences of 50 consecutive timesteps. We decrease the maximum size from 16000 to 10000 for the 3D visual navigation experiments in order to fit inside a single NVIDIA GeForce RTX 2080 GPU. For DREAM's exploitation policy, the replay buffer stores up to 100K experiences (60K for 3D visual navigation).

For DREAM, we additionally use per timestep exploration reward penalty $c = 0.01$, decoder and stochastic encoder variance $\rho^2 = 0.1$, and information bottleneck weight $\lambda = 1$. For the MiniWorld experiments, we use $c = 0$.

C Analysis

C.1 Consistency

We restate the consistency result of DREAM (Section 5.1) and prove it below.

Proposition 1. *Assume $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}_\mu, \mathcal{T}_\mu \rangle$ is ergodic for all problems $\mu \in \mathcal{M}$. Let $V^*(\mu)$ be the maximum expected returns achievable by any exploitation policy with access to the problem ID μ, i.e., with complete information. Let $\pi_\star^{task}$, $\pi_\star^{exp}$, $F_\star$ and $q_\star(z \mid \tau^{exp})$ be the optimizers of the* DREAM *objective. Then for long-enough exploration episodes and expressive-enough function classes,*

$$\mathbb{E}_{\mu \sim p(\mu), \tau^{exp} \sim \pi_\star^{exp}, z \sim q_\star(z \mid \tau^{exp})}\left[V^{\pi_\star^{task}}(z; \mu)\right] = \mathbb{E}_{\mu \sim p(\mu)}\left[V^*(\mu)\right].$$

Proof. Recall that $\pi_\star^{\text{task}}$ and $F_\star(z \mid \mu)$ are optimized with an information bottleneck according to Equation 2. Note that if $\pi_\star^{\text{task}}$ is optimized over an expressive enough function class and λ approaches 0, then $\pi_\star^{\text{task}}$ achieves the desired expected returns conditioned on the stochastic encoding of the problem $F_\star(z \mid \mu)$ (i.e., has complete information):

$$\mathbb{E}_{z \sim F_\star(z \mid \mu)}\left[V^{\pi_\star^{\text{task}}}(z; \mu)\right] = \mathbb{E}_{\mu \sim p(\mu)}\left[V^*(\mu)\right],$$

where $V^{\pi_\star^{\text{task}}}(z; \mu)$ is the expected returns of $\pi_\star^{\text{task}}$ on problem μ given embedding z. Therefore, it suffices to show that the distribution over z from the decoder $q_\star(z \mid \tau^{\text{exp}})$ is equal to the distribution from the encoder $F_\star(z \mid \mu)$ for all exploration trajectories in the support of $\pi^{\text{exp}}(\tau^{\text{exp}} \mid \mu)$[2], for each problem μ. Then,

$$\begin{aligned}\mathbb{E}_{\mu \sim p(\mu), \tau^{\text{exp}} \sim \pi_\star^{\text{exp}}, z \sim q_\star(z \mid \tau^{\text{exp}})}\left[V^{\pi_\star^{\text{task}}}(z; \mu)\right] &= \mathbb{E}_{z \sim F_\star(z \mid \mu)}\left[V^{\pi_\star^{\text{task}}}(z; \mu)\right] \\ &= \mathbb{E}_{\mu \sim p(\mu)}\left[V^*(\mu)\right]\end{aligned}$$

as desired. We show that this occurs as follows.

Given stochastic encoder $F_\star(z \mid \mu)$, exploration policy $\pi_\star^{\text{exp}}$ maximizes $I(\tau^{\text{exp}}; z) = H(z) - H(z \mid \tau^{\text{exp}})$ (Equation 3) by assumption. Since only $H(z \mid \tau^{\text{exp}})$ depends on $\pi_\star^{\text{exp}}$, the exploration policy outputs trajectories that minimize

$$\begin{aligned}H(z \mid \tau^{\text{exp}}) &= \mathbb{E}_{\mu \sim p(\mu)}\left[\mathbb{E}_{\tau^{\text{exp}} \sim \pi^{\text{exp}}(\tau^{\text{exp}} \sim \mu)}\left[\mathbb{E}_{z \sim F_\star(z \mid \mu)}\left[-\log p(z \mid \tau^{\text{exp}})\right]\right]\right] \\ &= \mathbb{E}_{\mu \sim p(\mu)}\left[\mathbb{E}_{\tau^{\text{exp}} \sim \pi^{\text{exp}}(\tau^{\text{exp}} \sim \mu)}\left[H(F_\star(z \mid \mu), p(z \mid \tau^{\text{exp}}))\right]\right],\end{aligned}$$

where $p(z \mid \tau^{\text{exp}})$ is the true conditional distribution and $H(F_\star(z \mid \mu), p(z \mid \tau^{\text{exp}}))$ is the cross-entropy. The cross-entropy is minimized when $p(z \mid \tau^{\text{exp}}) = F_\star(z \mid \mu)$, which can be achieved with long enough exploration trajectories T if $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}_\mu, \mathcal{T}_\mu \rangle$ is ergodic (by visiting each transition sufficiently many times). Optimized over an expressive enough function class, $q_\star(z \mid \tau^{\text{exp}})$ equals the true conditional distribution $p(z \mid \tau^{\text{exp}})$ at the optimum of Equation 3, which equals $F_\star(z \mid \mu)$ as desired. □

C.2 Tabular Example

We first formally detail a more general form of the simple tabular example in Section 5.2, where episodes are horizon H rather than 1-step bandit problems. Then we prove sample complexity bounds for RL^2 and DREAM, with ϵ-greedy tabular Q-learning with $\epsilon = 1$, i.e., uniform random exploration.

[2] We slightly abuse notation to use $\pi^{\text{exp}}(\tau^{\text{exp}} \mid \mu)$ to denote the distribution of exploration trajectories τ^{exp} from rolling out π^{exp} on problem μ.

Setting. We construct this horizon H setting so that taking a *sequence* of H actions $\mathbf{a}_\star$ (instead of a single action as before) in the exploration episode leads to a trajectory $\tau_\star^{\text{exp}}$ that reveals the problem μ; all other action sequences $\mathbf{a}$ lead to a trajectory $\tau_{\mathbf{a}}^{\text{exp}}$ that reveals no information. Similarly, the problem μ identifies a unique sequence of H actions $\mathbf{a}_\mu$ that receives reward 1 during exploitation, while all other action sequences receive reward 0. Again, taking the action sequence $\mathbf{a}_\star$ during exploration is therefore necessary and sufficient to obtain optimal reward 1 during exploitation.

We formally construct this setting by considering a family of episodic MDPs $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}_\mu, T_\mu \rangle$ parametrized by the problem ID $\mu \in \mathcal{M}$, where:

- Each exploitation and exploration episode is horizon H.
- The action space $\mathcal{A}$ consists of A discrete actions $\{1, 2, \dots, A\}$.
- The space of problems $\mathcal{M} = \{1, 2, \dots, |\mathcal{A}|^H\}$ and the distribution $p(\mu)$ is uniform.

To reveal the problem via the optimal action sequence $\mathbf{a}_\star$ and to allow $\mathbf{a}_\mu$ to uniquely receive reward, we construct the state space and deterministic dynamics as follows.

- States $s \in \mathcal{S}$ are $(H + 2)$-dimensional and the deterministic dynamics are constructed so the first index represents the current timestep t, the middle H indices represent the history of actions taken, and the last index reveals the problem ID if $\mathbf{a}_\star$ is taken. The initial state is the zero vector $s_0 = \mathbf{0}$ and we denote the state at the t-th timestep s_t as $(t, a_0, a_1, \dots, a_{t-1}, 0, \dots, 0, 0)$.
- The optimal exploration action sequence $\mathbf{a}_\star$ is set to be taking action 1 for H timesteps. In problem μ taking action $a_{H-1} = 1$ at state $s_{H-1} = (H - 1, a_0 = 1, \dots, a_{H-2} = 1, 0, 0)$ (i.e., taking the entire action sequence $\mathbf{a}_\star$) transitions to the state $s_H = (H, a_0 = 1, \dots, a_{H-2} = 1, a_{H-1} = 1, \mu)$, revealing the problem μ.
- The action sequence $\mathbf{a}_\mu$ identified by the problem μ is set as the problem μ in base $|\mathcal{A}|$: i.e., $\mathbf{a}_\mu$ is a sequence of H actions $(a_0, a_1, \dots, a_{H-1})$ with $\sum_{t=0}^{H-1} a_t |\mathcal{A}|^t = \mu$. In problem μ with $\mathbf{a}_\mu = (a_0, a_1, \dots, a_{H-1})$, taking action a_{H-1} at timestep $H - 1$ at state $s_{H-1} = (H - 1, a_0, a_1, \dots, a_{H-2}, 0, 0)$ (i.e., taking the entire action sequence $\mathbf{a}_\mu$) yields $\mathcal{R}_\mu(s_{H-1}, a_{H-1}) = 1$. Reward is 0 everywhere else: i.e., $\mathcal{R}_\mu(s, a) = 0$ for all other states s and actions a.
- With these dynamics, the exploration trajectory $\tau_{\mathbf{a}}^{\text{exp}} = (s_0, a_0, r_0, \dots, s_H)$ is uniquely identified by the action sequence $\mathbf{a}$ and the problem μ if revealed in s_H. We therefore write $\tau_{\mathbf{a}}^{\text{exp}} = (\mathbf{a}, \mu)$ for when $\mathbf{a} = \mathbf{a}_\star$ reveals the problem μ, and $\tau_{\mathbf{a}}^{\text{exp}} = (\mathbf{a}, 0)$, otherwise.

Uniform random exploration. In this general setting, we analyze the number of samples required to learn the optimal exploration policy with RL^2 and DREAM via ϵ-greedy tabular Q-learning. We formally analyze the simpler case where $\epsilon = 1$, i.e., uniform random exploration, but empirically find that DREAM only learns faster with smaller ϵ, and RL^2 only learns slower.

In this particular tabular example with deterministic dynamics that encode the entire action history and rewards, learning a per timestep Q-value is equivalent to learning a Q-value for the entire trajectory. Hence, we denote exploration Q-values $\hat{Q}^{\text{exp}}(\mathbf{a})$ estimating the returns from taking the entire sequence of H actions $\mathbf{a}$ at the initial state s_0 and exeuction Q-values $\hat{Q}^{\text{task}}(\tau^{\text{exp}}, \mathbf{a})$ estimating the returns from taking the entire sequence of H actions $\mathbf{a}$ at the initial state s_0 given the exploration trajectory τ^{exp}. We drop s_0 from notation, since it is fixed.

Recall that RL^2 learns exploration Q-values $\hat{Q}^{\text{exp}}$ by regressing toward the exploitation Q-values $\hat{Q}^{\text{task}}$. We estimate the exploitation Q-values $\hat{Q}^{\text{task}}(\tau^{\text{exp}}, \mathbf{a})$ as the sample mean of returns from taking actions $\mathbf{a}$ given the exploration trajectory τ^{exp} and estimate the exploration Q-values $\hat{Q}^{\text{exp}}(\mathbf{a})$ as the sample mean of the targets. More precisely, for action sequences $\mathbf{a} \neq \mathbf{a}_\star$, the resulting exploration trajectory $\tau_{\mathbf{a}}^{\text{exp}}$ is deterministically $(\mathbf{a}, 0)$, so we set $\hat{Q}^{\text{exp}}(\mathbf{a}) = \hat{V}^{\text{task}}(\tau_{\mathbf{a}}^{\text{exp}}) = \max_{\mathbf{a}'} \hat{Q}^{\text{task}}(\tau_{\mathbf{a}}^{\text{exp}}, \mathbf{a}')$. For $\mathbf{a}_\star$, the resulting exploration trajectory $\tau_{\mathbf{a}_\star}^{\text{exp}}$ may be any of $(\mathbf{a}_\star, \mu)$ for $\mu \in \mathcal{M}$, so we set $\hat{Q}^{\text{exp}}(\mathbf{a}_\star)$ as the empirical mean of $\hat{V}^{\text{task}}(\tau_{\mathbf{a}_\star}^{\text{exp}})$ of observed $\tau_{\mathbf{a}_\star}^{\text{exp}}$.

Recall that DREAM learns exploration Q-values $\hat{Q}^{\text{exp}}$ by regressing toward the learned decoder $\log \hat{q}(\mu \mid \tau_{\mathbf{a}}^{\text{exp}})$. We estimate the decoder $\hat{q}(\mu \mid \tau_{\mathbf{a}}^{\text{exp}})$ as the empirical counts of $(\mu, \tau_{\mathbf{a}}^{\text{exp}})$ divided by the empirical counts of $\tau_{\mathbf{a}}^{\text{exp}}$ and similarly estimate the Q-values as the empirical mean of $\log \hat{q}(\mu \mid \tau_{\mathbf{a}}^{\text{exp}})$. We denote the exploration Q-values learned after t timesteps as $\hat{Q}_t^{\text{exp}}$, and similarly denote the estimated decoder after t timesteps as $\hat{q}_t$.

Given this setup, we are ready to state the formal sample complexity results. Intuitively, learning the exploitation Q-values for RL^2 is slow, because, in problem μ, it involves observing the optimal exploration trajectory from taking actions $\mathbf{a}_\star$ and then observing the corresponding exploitation actions $\mathbf{a}_\mu$, which only jointly happens roughly once per $|\mathcal{A}|^{2H}$ samples. Since RL^2 regresses the exploration Q-values toward the exploitation Q-values, the exploration Q-values are also slow to learn. In contrast, learning the decoder $\hat{q}(\mu \mid \tau_{\mathbf{a}}^{\text{exp}})$ is much faster, as it is independent of the exploitation actions, and in particular, already learns the correct value from a single sample of $\mathbf{a}_\star$. We formalize this intuition in the following proposition, which shows that DREAM learns in a factor of at least $|\mathcal{A}|^H |\mathcal{M}|$ fewer samples than RL^2.

Proposition 2. *Let T be the number of samples from uniform random exploration such that the greedy-exploration policy is guaranteed to be optimal (i.e., $\arg\max_{\mathbf{a}} \hat{Q}_t^{exp}(\mathbf{a}) = \mathbf{a}_\star$) for all $t \geq T$. If $\hat{Q}^{exp}$ is learned with* DREAM*, the expected value of T is $\mathcal{O}(|\mathcal{A}|^H \log |\mathcal{A}|^H)$. If $\hat{Q}^{exp}$ is learned with RL^2, the expected value of T is $\Omega(|\mathcal{A}|^{2H} |\mathcal{M}| \log |\mathcal{A}|^H)$.*

Proof. For DREAM, $\hat{Q}_T^{\text{exp}}(\mathbf{a}_\star) > \hat{Q}_T^{\text{exp}}(\mathbf{a})$ for all $\mathbf{a} \neq \mathbf{a}_\star$ if $\log \hat{q}_T(\mu \mid (\mathbf{a}_\star, \mu)) > \log \hat{q}_T(\mu \mid (\mathbf{a}, 0))$ for all μ and $\mathbf{a} \neq \mathbf{a}_\star$. For all $t \geq T$, $\hat{Q}_t^{\text{exp}}$ is guaranteed to be optimal, since no sequence of samples will cause $\log \hat{q}_t(\mu \mid (\mathbf{a}_\star, \mu)) = 0 \leq \log \hat{q}_t(\mu \mid (\mathbf{a}, 0))$ for any $\mathbf{a} \neq \mathbf{a}_\star$. This occurs once we've observed $(\mu, (\mathbf{a}, 0))$ for two distinct μ for each $\mathbf{a} \neq \mathbf{a}_\star$ and we've observed $(\mu, (\mathbf{a}_\star, \mu))$ for at least one μ. We can compute an upperbound on the expected number of samples required to observe $(\mu, \tau_{\mathbf{a}}^{\text{exp}})$ for two distinct μ for each action sequence $\mathbf{a}$ by casting this as a coupon collector problem, where each pair $(\mu, \tau_{\mathbf{a}}^{\text{exp}})$ is a coupon. There are $2|\mathcal{A}|^H$ total coupons to collect. This yields that the expected number of samples is $\mathcal{O}(|\mathcal{A}|^H \log |\mathcal{A}|^H)$.

For RL^2, the exploration policy is optimal for all timesteps $t \geq T$ for some T only if the exploitation values $\hat{V}_T^{\text{task}}(\tau^{\text{exp}} = (\mathbf{a}_\star, \mu)) = 1$ for all μ in $\mathcal{M}$. Otherwise, there is a small, but non-zero probability that $\hat{V}_t^{\text{task}}(\tau^{\text{exp}} = (\mathbf{a}, 0))$ will be greater at some $t > T$. For the exploitation values to be optimal at all optimal exploration trajectories $\hat{V}_T^{\text{task}}(\tau^{\text{exp}} = (\mathbf{a}_\star, \mu)) = 1$ for all $\mu \in \mathcal{M}$, we must jointly observe exploration trajectory $\tau^{\text{exp}} = (\mathbf{a}_\star, \mu)$ and corresponding action sequence $\mathbf{a}_\mu$ for each problem $\mu \in \mathcal{M}$. We can lower bound the expected number of samples required to observe this by casting this as a coupon collector problem, where each pair $(\tau^{\text{exp}} = (\mathbf{a}_\star, \mu), \mathbf{a}_\mu)$ is a coupon. There are $|\mathcal{M}| \cdot |\mathcal{A}|^H$ unique coupons to collect and collecting any coupon only occurs with probability $\frac{1}{|\mathcal{A}|^H}$ in each episode. This yields that the expected number of samples is $\Omega(|\mathcal{A}|^{2H} \cdot |\mathcal{M}| \cdot \log |\mathcal{A}|^H)$. $\square$