Explore then Execute: Adapting without Rewards via Factorized Meta-Reinforcement Learning

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

Abstract

We seek to efficiently learn by leveraging shared structure between different tasks and environments. For example, cooking is similar in different kitchens, even though the ingredients may change location. In principle, meta-reinforcement learning approaches can exploit this shared structure, but in practice, they fail to adapt to new environments when adaptation requires targeted exploration (e.g., exploring the cabinets to find ingredients in a new kitchen). We show that existing approaches fail due to a chicken-and-egg problem: learning what to explore requires knowing what information is critical for solving the task, but learning to solve the task requires already gathering this information via exploration. For example, exploring to find the ingredients only helps a robot prepare a meal if it already knows how to cook, but the robot can only learn to cook if it already knows where the ingredients are. To address this, we propose a new exploration objective (DREAM), based on identifying key information in the environment, independent of how this information will exactly be used solve the task. By decoupling exploration from task execution, DREAM explores and consequently adapts to new environments, requiring *no reward signal* when the task is specified via an instruction. Empirically, DREAM scales to more complex problems, such as sparse-reward 3D visual navigation, while existing approaches fail from insufficient exploration.[1]

1 Introduction

A general-purpose agent should be able to perform multiple tasks across multiple environments, both of which share considerable structure. 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 first explore to discover the locations of the ingredients, and then use this information to cook. Existing meta-RL methods have shown promise for adapting to new tasks and environments, but, as we identify in this work, are poorly equipped when adaptation requiring sophisticated 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 similar environment and task. It is allowed to gather experience in a few initial (training) episodes, and its goal is to then maximize returns on all subsequent (testing) episodes. A common meta-RL approach is to train a recurrent neural network (RNN) [7, 38, 33, 44, 16, 18] to maximize test returns on each meta-training environment and task. With enough capacity, such approaches can express the optimal adaptation strategy, which involves both *exploring* in the initial few episodes (to find relevant information) and then *executing* in the subsequent episodes (use the gained information to solve the task) during meta-testing. However, optimizing for exploration and execution end-to-end in this way creates a challenging chicken-and-egg optimization problem, leading to bad local

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

Preprint. Under review.

optima: Learning how to explore requires knowing what information is critical for actually solving the task, but learning to solve the task requires already gathering this information via exploration. As a concrete example, exploring to locate the ingredients only helps a robot prepare a meal if it already knows how to cook, but the robot can only learn to cook if it already knows where the ingredients are.

The key insight of this work is to construct a decoupled exploration objective by first automatically identifying task-relevant information and then training an exploration policy to uncover this information. Concretely, we assume access to a problem ID (during meta-training but not meta-testing) that identifies the task and environment, which may contain information irrelevant to solving the task (e.g., the color of the walls for a cooking task). We try to learn a representation of the problem ID that discards this irrelevant information by simultaneously minimizing the mutual information between the ID representation and the ID itself, and learning an execution policy to solve tasks conditioned on this ID representation. Then, we train an exploration policy to produce trajectories that contain precisely the information in the learned ID representation, ensuring efficient and targeted exploration (Section 4).

Beyond the above optimization challenges, a second issue that prevents existing approaches from exploring to identify relevant information in the environment relates to the problem formulation rather than the approaches themselves. In the standard meta-RL formulation, both the task and environment vary across problems and must be inferred via trial-and-error. While the environment must naturally be inferred through interaction, hiding *all* information about the task, e.g., the meal to cook (or the velocity to run at [10]) is unrealistic and can create unnecessary exploration challenges, particularly in sparse reward settings. For example, in the standard meta-RL problem formulation, the robot would inefficiently need to cook various meals until it guesses the correct meal to cook. Instead, we propose a setting called instruction-based meta-RL (IMRL), where the agent is provided information about the task via *instructions* (e.g., "cook a pizza" or a one-hot representation). Additionally, while in the standard meta-RL problem formulation, the agent requires reward observations to infer the task, by providing instructions that specify the task, IMRL enables *reward-free adaptation*, where the agent learns from rewards during meta-training as usual, but executes instructions in a new environment without ever receiving rewards in that environment during meta-testing (Section 3).

Overall, we present two contributions: (i) we identify the problem of coupling between exploration and execution and provide an approach that overcomes this via a decoupled objective, and (ii) we propose a new meta-RL setting (IMRL), that can enable reward-free adaptation. When applied to IMRL, our decoupled approach, called DREAM (**D**ecoupling **R**eward-free **E**xplor**A**tion and execution in **M**eta-RL) learns sophisticated exploration strategies and enables reward-free adaptation on a sparse-reward 3D visual navigation problem, achieving near-optimal performance. In contrast, existing state-of-the-art meta-RL approaches (IMPORT, VARIBAD, RL^2) do not learn the optimal exploration strategy and achieve zero reward, even when they receive reward observations for adaptation. Furthermore, since the problem of coupling is not specific to IMRL, DREAM also achieves higher returns than these existing state-of-the-art approaches in the standard meta-RL setting, as well (Section 6).

2 Preliminaries

Meta-Reinforcement Learning Setting. The standard 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, parametrized by a *problem* $\mu \in \mathcal{M}$. Colloquially, we refer to the dynamics as the *environment*, the rewards as the *task*, and the entire MDP as the *problem*. Different problems share structure (e.g., similar kitchens or cooking similar meals), and are drawn from some distribution $\mu \sim p(\mu)$. Following prior problem settings [10, 26, 28, 9] and terminology from Duan et al. [7], we define a *trial* as $N + 1$ episodes in the same MDP, with an initial training (exploration) episode of T steps, typically used to infer dynamics and rewards, followed by N testing (execution) episodes. To enable efficient exploration, we allow the agent to take a special "end-episode" action a_{term} to terminate exploration early. Meta-training and meta-testing both consist of sampling a problem $\mu \sim p(\mu)$ and running a trial. The agent's goal is to maximize the returns summed over the N execution episodes during meta-testing. Following prior work [26, 16, 18], we make the problem ID of μ available for meta-training, but not meta-testing, when the agent must infer this via exploration.

For convenience, we formally express the objective in terms of an exploration policy π^{exp} used in the exploration episode and an execution policy π^{exec} used in execution episodes, but these policies may be the same or share parameters. The execution policy π^{exec} may depend on all prior experiences in the trial, including the exploration trajectory $\tau^{\text{exp}} = (s_0, a_0, r_0, \ldots, s_T) \sim \pi^{\text{exp}}$ from rolling out π^{exp}

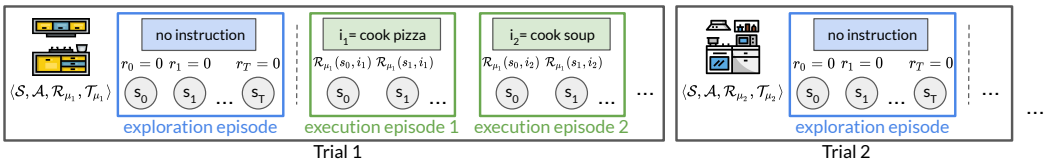

Figure 1: *Reward-free* instruction-based meta-RL: Meta-training trials (in a single problem) consist of a reward-free exploration episode followed by many execution episodes with rewards defined by instructions.

in the exploration episode. The goal of the agent is therefore to maximize:

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

where $V^{\pi^{\text{exec}}}(\tau^{\text{exp}}; \mu)$ is the expected return of π^{exec} in problem μ in the execution episodes.

Learning to Reinforcement Learn. A common meta-RL approach is the *Learning to Reinforcement Learn* framework [38, 7, 44, 18, 16], which directly maximizes the objective $\mathcal{J}$ in (1) by learning a single recurrent policy $\pi(a_t \mid s_t, \tau_{:t})$ for both exploration and execution (i.e., $\pi^{\text{exec}} = \pi^{\text{exp}} = \pi$). This policy 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, the recurrent representation of $\tau_{:t}$ can encode uncertainty over which problem the agent is currently in, i.e., a posterior over problems $p(\mu \mid \tau_{:t})$ [17, 44]. The optimal policy explores to reduce this uncertainty in the initial exploration episode, ideally until the the same actions lead to high returns on all potential problems the agent might be in. By optimizing both exploration and execution end-to-end to maximize returns, this approach can learn the optimal policy, but optimization is challenging, which we show in Section 4.1. We refer to this framework by its canonical instantiation, RL^2 [7].

3 Instruction-based Meta-RL

While the standard meta-RL setting (Section 2) captures adapting to new problems, it requires exploring to obtain both key information about the environment (the ingredients' location in a kitchen), as well as the task itself (the meal to cook). It is often more realistic and efficient to directly convey the task to the agent. We therefore propose a new meta-RL setting called instruction-based meta-RL (IMRL), where the agent receives *instructions* conveying the task. Additionally, unlike the standard setting, where an agent repeatedly executes the same task (e.g., cook pizza) in a trial, we allow each episode to have a different task provided via a different instruction (e.g., cook pizza, then pasta, then soup), while the environment remains fixed. This forces exploration to capture shared structure between different tasks in the same environment, which the standard setting does not. Specifically, we augment the state with an *instruction* $i \in \mathcal{I}$ (e.g., cook pizza), represented in this work as collections of one-hots, but in other works could be represented with natural language or images (e.g., of a desired meal). On each episode, we sample an instruction i from a distribution of instructions $p_\mu(i)$ and the agent receives the rewards $\mathcal{R}_\mu(s_t, i)$ at each timestep t in problem μ. Note that when the instruction is always the same, (e.g., $|\mathcal{I}| = 1$), IMRL can recover the standard setting, so any algorithms developed for IMRL (e.g., Section 4) also apply to the standard setting. We summarize IMRL in Figure 1 for a special case described below, where we remove instructions and rewards from exploration episodes.

Special case: reward-free adaptation. When the instructions i together with the transition observations, i.e., (s, a, s')-tuples, contain enough information about the reward function, executing instructions does not require observing the rewards directly during exploration. For example, a robot can execute the instruction "cook the soup recipe on the fridge" by reading and following the recipe on the fridge, without needing to observe the corresponding rewards. Consequently, we optionally make the exploration episode *reward-free*, providing neither instruction nor rewards. Further, if the agent does not adapt using reward feedback during execution episodes, the entire process of adaptation at meta-test time can be made reward-free: Specifically, during meta-training, the exploration episode is reward-free, but the agent receives rewards during execution episodes. During meta-testing, the agent must adapt to the new task and environment without any reward observations. This models real-world situations where providing rewards is expensive (e.g., a robot chef would ideally adapt to a new home kitchen without any human supervision). IMRL allows us to conveniently model this special case.

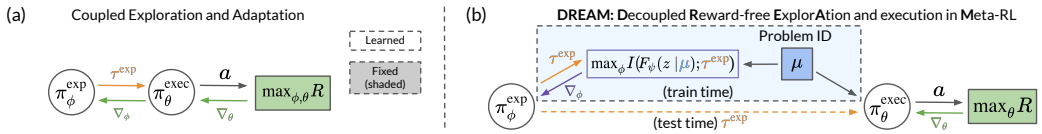

Figure 2: (a) Coupling between the exploration policy π^{exp} and execution policy π^{exec}. The exploration and execution policies are illustrated separately for clarity, but may be a single policy. Since the two policies depend on each other (for gradient signal and the τ^{exp} distribution), it is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

4 Decoupling Exploration and Execution

4.1 The Problem with Coupling Exploration and Execution

We begin by showing how approaches like RL^2, which optimize exploration and execution episodes end-to-end by maximizing $\mathcal{J}$ in (1), can be sample inefficient in IMRL (and therefore in the standard meta-RL setting as well). For clarity, we refer to the policy followed in exploration episodes as π^{exp} and the policy followed in execution episodes as π^{exec}, although for RL^2 they are the same policy.

Figure 2a illustrates this end-to-end training approach. Note that π^{exec} relies on π^{exp} for good exploration data in the form of τ^{exp} in order to learn to solve the task. Additionally, note that learning π^{exp} relies on gradients passed through π^{exec}. If π^{exec} cannot effectively solve the task, then these gradients will be uninformative. This causes a bad local optimum: if our current (suboptimal) π^{exec} obtains low rewards with a good informative trajectory $\bar{\tau}^{\text{exp}}$, the low reward would cause π^{exp} to learn to *not* generate $\bar{\tau}^{\text{exp}}$. This causes π^{exp} to instead generate trajectories $\underline{\tau}^{\text{exp}}$ that lack information required to obtain high reward, further preventing the execution policy π^{exec} from learning. Typically, early in training, both π^{exp} and π^{exec} are stuck in this local optimum, where neither policy can become optimal without many samples. We illustrate this effect in a simple example in Section 5.2.

4.2 DREAM: Decoupling Reward-free Exploration and Execution in Meta-learning

The key idea behind DREAM is to sidestep this local optimum by optimizing decoupled objectives for the exploration and execution policies. Intuitively, the goal of the exploration policy is to identify the distinguishing characteristics of the environment and task relevant to executing instructions. An initial approach is to train π^{exp} so that it produces trajectories that are predictive of the problem ID or dynamics [43]. This yields a policy that can distinguish problems, but inefficiently explores instruction-irrelevant attributes, such as the color of the walls. To avoid this, we propose to derive a stochastic problem encoding $F_\psi(z \mid \mu)$, which extracts only the information necessary to execute instructions in the problem μ. DREAM obtains this encoder by training an execution policy π^{exec} conditioned on the encoder's outputs, with an information bottleneck on z. Then, DREAM trains an exploration policy π^{exp} to produce trajectories with high mutual information with z. In this approach, the execution policy π^{exec} no longer relies on effective exploration π^{exp} to learn, and once $F_\psi(z \mid \mu)$ is learned, the the exploration policy π^{exp} can also learn independently from π^{exec}, decoupling the two optimization processes. Later, during meta-testing, when the problem ID μ is unavailable, the two policies easily combine, since the trajectories generated by π^{exp} contain the same information as the stochastic encodings $F_\psi(z \mid \mu)$ that the execution policy π^{exec} trained on (overview in Figure 2b).

Learning the problem ID encodings. We begin with learning a good stochastic encoder $F_\psi(z \mid \mu)$ parametrized by ψ and execution policy $\pi_\theta^{\text{exec}}(a \mid s, i, z)$ parameterized by θ. Unlike RL^2, we choose not to make π_θ^{exec} recurrent for simplicity, relying on z to contain the necessary information. We learn $F_\psi(z \mid \mu)$ jointly with the execution policy $\pi_\theta^{\text{exec}}(a \mid s, i, z)$ by optimizing the following objective:

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

where $V^{\pi_\theta^{\text{exec}}}(i, z; \mu)$ is the expected return of π_θ^{exec} on problem μ given instruction i and encoding z. Importantly, both terms are independent of the exploration policy π^{exp}.

We minimize the mutual information $I(z;\mu)$ by applying a variational upper bound [3] as follows.

$$I(z;\mu) = \mathbb{E}_\mu\left[\mathrm{D}_{\text{KL}}(F_\psi(z \mid \mu)||r(z))\right] - \mathrm{D}_{\text{KL}}(p_\psi(z)||r(z)) \le \mathbb{E}_\mu\left[\mathrm{D}_{\text{KL}}(F_\psi(z \mid \mu)||r(z))\right], \tag{3}$$

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 information required to optimally execute instructions, we can optimize the exploration policy π^{exp} to produce trajectories that encode this same information by maximizing their mutual information. 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 optimized by applying a variational lower bound [3] as follows.

$$I(\tau^{\text{exp}}; z) = H(z) - H(z \mid \tau^{\text{exp}}) \geq H(z) + \mathbb{E}_{\mu, z \sim F_\psi, \tau^{\text{exp}} \sim \pi^{\text{exp}}} \left[\log q_\omega(z \mid \tau^{\text{exp}})\right] \tag{4}$$

$$= H(z) + \mathbb{E}_{\mu, z \sim F_\psi}[\log q_\omega(z)] + \mathbb{E}_{\mu, z \sim F_\psi, \tau^{\text{exp}} \sim \pi^{\text{exp}}} \left[\sum_{t=1}^{T} \log q_\omega(z \mid \tau^{\text{exp}}_{:t}) - \log q_\omega(z \mid \tau^{\text{exp}}_{:t-1})\right],$$

where q_ω is any distribution parameterized 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 serves as a decoder that generates the encoding z from the exploration trajectory τ^{exp}.

Note that only the third term depends on the exploration trajectory. Hence, we maximize this by training the exploration policy on rewards set to be the information gain:

$$r^{\text{exp}}_t(a_t, s_{t+1}, \tau^{\text{exp}}_{t-1}; \mu) = \mathbb{E}_{z \sim F_\psi(z \mid \mu)} \left[\log q_\omega(z \mid [s_{t+1}; a_t; \tau^{\text{exp}}_{:t-1}]) - \log q_\omega(z \mid \tau^{\text{exp}}_{:t-1})\right] - c. \tag{5}$$

Intuitively, the 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 $\tau^{\text{exp}}_{:t-1}$. The reward also includes a small per timestep penalty c to encourage exploring efficiently in as few timesteps as possible.

This reward is attractive because (i) it is independent from the execution policy and hence avoids the local optima highlighted in Section 4.1, and (ii) it is a dense reward signal and helps with credit assignment. Note that it is not Markovian, because it depends on τ^{exp}, so we learn a recurrent $\pi^{\text{exp}}_\phi(a_t \mid s_t, \tau^{\text{exp}}_{:t})$ parametrized by ϕ that conditions on its history $\tau^{\text{exp}}_{:t} = (s_0, a_0, r_0, \ldots, s_{t-1}, a_{t-1}, r_{t-1})$. For the reward-free exploration case, we simply omit the rewards r_t from τ^{exp}.

4.3 A Practical Implementation of DREAM

Altogether, DREAM learns four separate components (neural networks), which we detail below.

1. Encoder $F_\psi(z \mid \mu)$: For simplicity, we parameterize 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 (Equation 3) take the form of simple ℓ_2-regularization $\|f_\psi(\mu)\|_2^2$.
2. Decoder $q_\omega(z \mid \tau^{\text{exp}})$: Similarly, we parameterize 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 4), 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 5).
3. Execution policy π^{exec}_θ and 4. Exploration policy π^{exp}_ϕ: We represent both policies as Q-networks and apply double deep Q-learning (DDQN) [37], treating (s, i, z) as the state for π^{exec}_θ.

While the above suggests training the encoder and execution policy to convergence, and then using these to train the exploration policy, we find it less cumbersome to learn all components together in an EM-like fashion where in the exploration episode, we assume f_ψ and π^{exec}_θ are fixed. We also train π^{exec}_θ conditioned on the exploration trajectory $g_\omega(\tau^{\text{exp}})$, instead of exclusively training on the outputs of the encoder $z \sim F_\psi(z \mid \mu)$. We include all details and a summary (Algorithm 1) in Appendix A.

5 Analysis

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 works [43, 26, 12, 13], which also decouple exploration from execution, but do not recover the optimal policy even with infinite data. Formally,

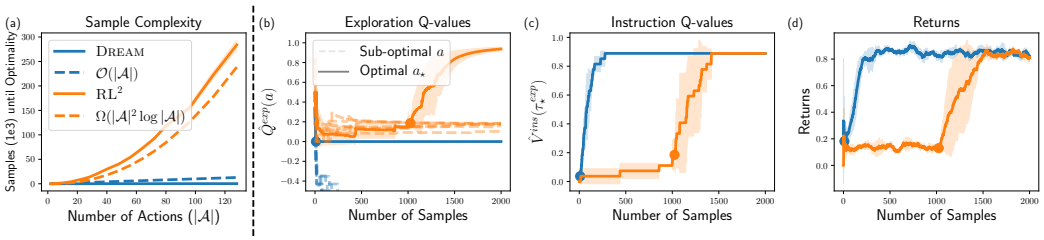

Figure 3: (a) Sample complexity of learning the optimal exploration policy as the action space $|\mathcal{A}|$ grows (1000 seeds). (b) Exploration Q-values $\hat{Q}^{\text{exp}}(a)$. The policy $\arg\max_a \hat{Q}^{\text{exp}}(a)$ is optimal after the dot; (c) instruction values given optimal trajectory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

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 $V^*(i;\mu)$ be the maximum expected returns achievable by any execution policy with access to μ on instruction i and problem μ, i.e., with complete information. Let $\pi_\star^{exec}, \pi_\star^{exp}, F_\star$ and $q_\star(z \mid \tau^{exp})$ be the optimizers of the* DREAM *objective. Then for large enough T (the length of the exploration episode) and expressive enough function classes,*

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

Optimizing the decoupled objective in DREAM achieves the maximal returns $V^*(i;\mu)$ even without access to μ during meta-testing, which we prove in Appendix C.1. DREAM requires ergodicity to guarantee achieving optimal returns, but we can remove this assumption by increasing the number of exploration episodes, and in practice, DREAM performs well on non-ergodic MDPs in our experiments.

5.2 Analysis of the Sample Complexity of Coupled and Decoupled Approaches

With enough capacity, approaches like RL^2 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 $\tau_\star^{\text{exp}}$ that reveals the problem μ; all other actions a reveal no information and lead to τ_a^{exp}. The problem μ identifies a unique action that receives reward 1 during execution; 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 and DREAM to learn the optimal exploration policy with ϵ-greedy tabular Q-learning. We precisely describe a more general 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 execution Q-values form targets for the exploration Q-values: $\hat{Q}^{\text{exp}}(a) \leftarrow \hat{V}^{\text{exec}}(\tau_a^{\text{exp}}) := \max_{a'} \hat{Q}^{\text{exec}}(\tau_a^{\text{exp}}, a')$. We drop the fixed initial state from notation. This creates the local optimum in Section 4.1. Initially $\hat{V}^{\text{exec}}(\tau_\star^{\text{exp}})$ is low, as the execution 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 $\hat{V}^{\text{exec}}(\tau_\star^{\text{exp}})$ 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 execution 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 $\hat{Q}^{\text{exec}}$, since it does not depend on the execution actions. Consequently, DREAM's exploration policy quickly becomes optimal (dot in Figure 3b), which allows quickly learning the execution Q-values and achieving high reward (Figures 3c and 3d).

6 Experiments

Many popular meta-RL benchmarks are not designed for and do not test exploration. For example, in the benchmark used by Finn et al. [10], Rothfuss et al. [28], Rakelly et al. [26], Fakoor et al. [9], many tasks can be solved with only a few timesteps of almost *any* exploration. Therefore, we design several

didactic benchmarks (focusing on the reward-free adaptation special case of IMRL with instructions) to stress test various aspects of exploration in meta-RL: if DREAM can (i) efficiently explore, even in presence of distractions; (ii) leverage objects (e.g., a map) to aid exploration; (iii) learn exploration and execution behaviors that generalize to unseen problems and instructions; (iv) efficiently explore in the standard meta-RL setting (with rewards during exploration and without instructions). We also evaluate on a challenging sparse-reward 3D visual navigation benchmark requiring sophisticated exploration.

As we expand upon in Section 7, existing meta-RL algorithms fall into two main categories (i) *end-to-end* approaches that optimize exploration and execution behaviors with a recurrent policy end-to-end based on execution rewards and (ii) *decoupled* approaches that use separate objectives for exploration and execution. We compare DREAM with state-of-the-art approaches in each category, namely:

1. RL^2 [7, 38]: the canonical end-to-end approach described in Section 2.
2. VARIBAD [44]: an end-to-end approach, which aims to optimally trade off between exploration and exploitation with an auxiliary loss to predict the dynamics and rewards from the recurrent state.
3. IMPORT [18]: an end-to-end-approach, which aims to improve sample complexity by (i) alternating between training the policy conditioned on the problem ID and its recurrent state and (ii) optimizing an auxiliary loss to minimize the distance between the recurrent state and problem ID embedding.
4. PEARL-UB: the analytically computed expected rewards achieved by optimal Thompson sampling exploration, assuming access to the optimal problem-specific policy and true posterior problem distribution. This is an upperbound on the final performance of PEARL [26], a decoupled approach.

We report the average returns achieved by each approach in trials with one exploration and one execution episode, averaged over 3 seeds with 1-std error bars. Unless otherwise specified, these trials are IMRL trials with reward-free exploration (full details in Appendix B).

6.1 Didactic Experiments

We first evaluate on grid world domains illustrated 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), and a one-hot indicator of the agent's inventory (none or an ingredient). 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, places the ingredients of the fridge into the agent's inventory; and *drop*, which, at the pot, places the agent's inventory into the pot. Episodes consist of 20 timesteps and the agent receives a reward of -0.1 at each timestep until the instruction, described below, is executed (full details in Appendix B.1 and qualitative results in Appendix B.2).

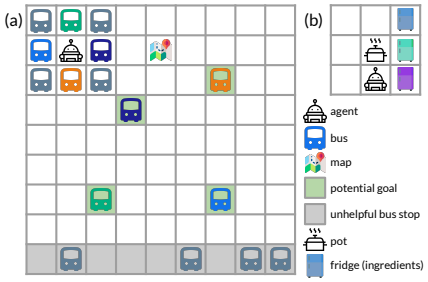

Figure 4: (a) Navigation. (b) Cooking.

Targeted exploration. We first test if these methods can efficiently explore in the presence of distractions in two versions of the family of problems in Figure 4a: *distracting bus* and *map*. In both, there are 4 possible instructions (corresponding to the 4 green potential goal locations). During each execution episode, 1 of these 4 instructions is randomly sampled and reaching the corresponding goal yields +1 reward and ends the episode. The 4 colored buses each lead to 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 execute any instruction. In different problems, the gray buses lead to different locations in the bottom row. 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.

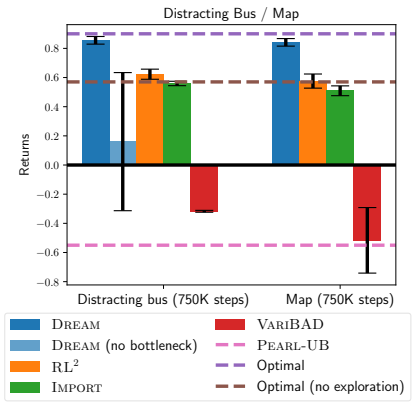

Figure 5: Navigation results. Only DREAM optimally explores all buses and the map.

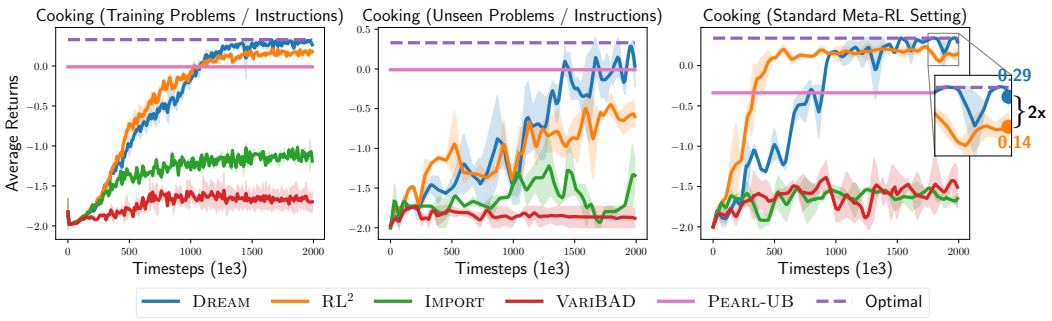

Figure 6: Cooking results. Only DREAM achieves optimal reward on training problems and instructions (left), on generalizing to unseen and problems and instructions (middle), and on the standard meta-RL setting (right).

Figure 5 summarizes the results. DREAM learns to optimally explore and thus receives optimal reward in both versions: In *distracting bus*, it ignores the unhelpful gray buses and rides 3 of the 4 helpful colored buses during exploration, which allows it to infer the destination of the 4th helpful bus as well. In *map*, it visits the map and ends the exploration episode. During execution, DREAM immediately reaches the goal by riding the correct colored bus. In contrast, indicative of the coupling problem (Section 4.1), IMPORT and RL^2 rarely explore buses during exploration and then consequently perform suboptimal exploitation by just walking to the goal, which achieves the same returns as no exploration at all. VARIBAD learns slower, likely from learning the dynamics in its auxiliary loss, but eventually matches the sub-optimal execution of IMPORT and RL^2 in ~3x as many samples (not shown).

Thompson sampling explores by sampling a problem from its posterior and then executes its policy conditioned on this problem. Since for any given problem (bus configuration) and instructions (goal), the optimal problem-specific policy rides the one bus leading to the goal, Thompson sampling 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. Therefore, PEARL cannot achieve optimal reward, even with infinite meta-training data, shown by the line for PEARL-UB.

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 gets lower reward.

Generalization to new problems / instructions. We test generalization to unseen problems / instructions in a cooking domain (Figure 4b). The fridges on the right each contain 1 of 7 different (color-coded) ingredients, determined by the problem. The fridges' contents are unobserved until the agent uses the "pickup" action at the fridge. Instructions specify placing 2 correct ingredients in the pot in the right order. We hold out ~1% of the problems and ~5% of the instructions during training.

Figure 6 shows the results on the training (left) and held-out (middle) problems and instructions. Only DREAM achieves near-optimal returns on both. During exploration, it investigates each fridge by using the "pick up" action, and then directly retrieves the correct ingredients during execution. RL^2 only sometimes explores the fridges during training, leading to lower returns. It also overfits to the training recipes and fails to cook the testing recipes. Training with the problem ID actually hurts IMPORT compared to RL^2: IMPORT successfully cooks conditioned on the problem ID, but fails on many recipes conditioned on its recurrent state, likely since the problem ID embeddings and recurrent state empirically become far apart. As before, VARIBAD learns slowly and Thompson sampling (PEARL-UB) cannot learn optimal exploration.

Standard meta-RL setting. While we focus on IMRL due to its realism, the coupling problem we highlight is not specific to IMRL, and also occurs in the standard meta-RL setting. We therefore also evaluate on a variant of the above cooking domain in the standard meta-RL setting, where there are no instructions, and the meal to cook must be inferred through reward observations during the exploration episode. The results are summarized in Figure 6 (right). Only DREAM optimally solves the task, achieving ~2x the reward of the next best approach, RL^2, which quickly gets stuck in a local optimum that it fails to escape, indicative of the coupling problem.

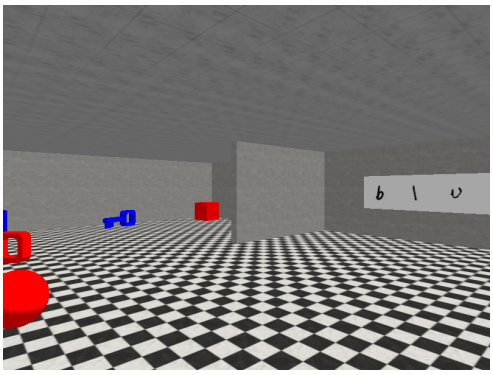

Figure 7: 3D Visual Navigation: the agent must read the sign to determine what colored object to go to.

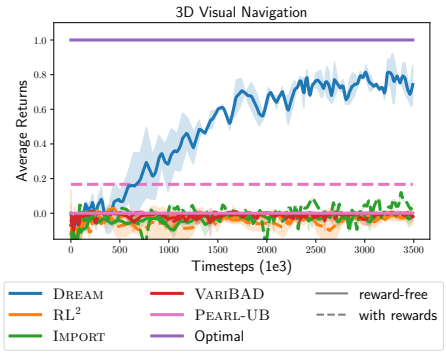

Figure 8: 3D visual navigation results. Only DREAM reads the sign and solves the task.

6.2 Sparse-Reward 3D Visual Navigation

We conclude with a variant of the problem family introduced by Kamienny et al. [18] with visual observations, which we make even more challenging by including more objects and a visual sign, illustrated in (Figure 7): There are two problems, one where the sign on the right says "blue" and the other where it says "red." The instructions specify whether the agent should go to the key, block, or ball, and the agent receives +1 reward for going to the object with the correct color (specified by the sign) and shape, and -1 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 images and its actions are to rotate left or right, move forward, or end the episode.

DREAM is the only method that learns to read the sign and achieve reward (Figure 8). Consistent with the results in Kamienny et al. [18], the other methods do not read the sign and consequently stay away from all the objects, in fear of receiving negative reward. This occurs even when we allow these methods to receive an instruction and rewards during the exploration episode (dashed lines), while DREAM successfully adapts *reward-free*.

7 Related Work

We draw on the long line of work on learning to adapt to new similar tasks [32, 35, 25, 5, 4, 14, 2, 31]. Within meta-RL, many works focus on adapting efficiently to a new task from few samples without optimizing the sample collection process, via updating the policy parameters [10, 28, 1, 41, 21], learning a model [24, 30], multi-task learning [9], or leveraging demonstrations [42]. Instead, we focus on problems where targeted exploration is critical for few-shot adaptation.

More closely related to our work are approaches that specifically explore to obtain the most informative samples. These fall into two main categories: end-to-end and decoupled approaches. The first category jointly learns exploration and adaptation behaviors end-to-end by training a recurrent policy to maximize returns [7, 38, 22, 33, 44, 16, 18, 27] and can represent the optimal policy [17], but, as this work shows, suffer from coupling between exploration and adaptation. Notably, many of these works use separate objectives to train the *representation* of explored transitions but not for directly improving the exploration behavior. As a result, the coupling problem remains. The second category decouples exploration from adaptation via, e.g., Thompson-sampling (TS) [34, 26], obtaining exploration trajectories predictive of dynamics or rewards [43, 13], or exploration noise [12]. These avoid the problem of coupling in the first category, but no longer learn optimal exploration. In particular, TS [26] explores by guessing the task and executing a policy for that task, and therefore cannot represent exploration behaviors that are different from adaptation [29]. Predicting the dynamics [43, 13] is inefficient when only a small subset of the dynamics are relevant to solving the task. In contrast, DREAM's objective is consistent and yields optimal exploration when maximized. Past work [11, 15, 8, 40] also optimizes mutual information objectives, but not for meta-RL.

IMRL also draws inspiration from prior work. Reward-free adaptation relates to prior work that considers ignoring the rewards in the first episodes [33] and when rewards are unavailable at test time [41]. Other works [24, 30] also convey the reward function to the agent, similar to instructions, though unlike IMRL, they allow the agent to query this reward function arbitrarily for planning.

8 Conclusion

In summary, we present two contributions. First, we identify that existing end-to-end meta-RL approaches suffer from a chicken-and-egg coupling problem, where learning good execution requires already having learned good exploration and vice-versa. Since typically neither exploration nor execution is good at the beginning of training, these approaches become stuck in local optima. We propose to break this cyclic dependency between learning exploration and execution with decoupled objectives (DREAM). Unlike prior decoupled objectives [43, 26, 12, 13] that do not learn the optimal exploration strategy, maximizing our decoupled objectives also maximizes returns. Second, we propose a more realistic meta-RL setting (IMRL) that (i) avoids unnecessary exploration for identifying the *task* by providing instructions to the agent and (ii) exploits shared structure across different tasks in the same environment. These two contributions are complementary, as IMRL enables reward-free adaptation in principle, and DREAM achieves this in practice.

We emphasize that the chicken-and-egg coupling problem and therefore, DREAM, applies more generally to any setting where some information is provided to the agent, and the rest must be discovered via exploration. Our experiments specifically show that DREAM achieves higher reward than state-of-the-art meta-RL methods in two such settings, IMRL, where the task is (partially) provided to the agent via instructions, and the standard meta-RL setting, where everything must be discovered by exploration. Other settings may also be of interest, such as providing the agent with some information about the environment (e.g., locations of *some* ingredients).

While our work provides many benefits, we study the setting where the agent is allowed initial exploration episodes. Other work, like VARIBAD, can in principle, adapt even without these initial episodes, which we leave to future work. Future work could also explore more sophisticated representations for the problem IDs and instructions, such as natural language.

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. 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] 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.

[8] 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.

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

[10] 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.

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

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] S. Ritter, J. X. Wang, Z. Kurth-Nelson, S. M. Jayakumar, C. Blundell, R. Pascanu, and M. Botvinick. Been there, done that: Meta-learning with episodic recall. *arXiv preprint arXiv:1805.09692*, 2018.

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

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

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

[31] 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.

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

[33] 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.

[34] 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.

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

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

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

[44] 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, parametrizing the policies as deep dueling double-Q networks [39, 37], with exploration Q-values $\hat{Q}^{\text{exp}}(s, \tau^{\text{exp}}, a; \phi)$ parametrized by ϕ (and target network parameters ϕ') and execution Q-values $\hat{Q}^{\text{exec}}(s, i, z, a; \theta)$ parametrized by θ (and target network parameters θ'). We train on trials with one exploration and one execution episode, but can test on arbitrarily many execution episodes, as the execution 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 sample an instruction and roll-out the execution policy, adding the roll-out to a separate replay buffer (lines 10-13). We train the execution 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 14-16). Concretely, we minimize the following loss function w.r.t., the parameters ϕ:

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

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

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

We perform a similar update with the execution Q-value parameters (lines 17-19). We sample $(s, a, r, s', i, \mu, \tau^{\text{exp}})$-tuples from the execution 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:

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

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

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

Finally, from the same execution replay buffer samples, we also update the problem ID embedder to enforce the information bottleneck (line 20) and the decoder to approximate the true conditional distribution (line 21) 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:

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

As is usual with deep Q-learning, 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 22-23).

B Experiment Details

B.1 Problem Details

Distracting bus / map. Riding each of the four colored buses teleports the agent to one of the green goal locations. In different problems, the destinations of the colored buses change, but the

Algorithm 1 DREAM DDQN

Initialize execution replay buffer $\mathcal{B}_{\text{exec}} = \{\}$ and exploration replay buffer $\mathcal{B}_{\text{exp}} = \{\}$
Initialize execution Q-value $\hat{Q}^{\text{exec}}$ parameters θ and target network parameters θ'
Initialize exploration Q-value $\hat{Q}^{\text{exp}}$ parameters ϕ and target network parameters ϕ'
Initialize problem ID embedder f_ψ parameters ψ and target parameters ψ'
Initialize trajectory embedder g_ω parameters ω and target parameters ω'
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 ϵ-greedy exploration policy $\hat{Q}^{\text{exp}}(s_t, \tau^{\text{exp}}_{:t}, a_t; \phi)$, producing trajectory $\tau^{\text{exp}} = (s_0, a_0, \ldots, s_T)$.
  Add tuples to the exploration replay buffer $\mathcal{B}_{\text{exp}} = \mathcal{B}_{\text{exp}} \cup \{(s_t, a_t, s_{t+1}, \mu, \tau^{\text{exp}})\}_t$.

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

  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 $r^{\text{exp}}_t = \|f_\psi(\mu) - g_\omega(\tau^{\text{exp}}_{:t})\|_2^2 - \left\|f_\psi(\mu) - g_\omega(\tau^{\text{exp}}_{:t-1})\right\|_2^2 - c$ (Equation 5).
  Optimize ϕ with DDQN update with tuple $(s_t, a_t, r^{\text{exp}}_t, s_{t+1})$

  Sample batches of $(s, a, r, s', i, \mu, \tau^{\text{exp}}) \sim \mathcal{B}_{\text{exec}}$ from execution replay buffer.
  Optimize θ and ω with DDQN update with tuple $((s, i, \tau^{\text{exp}}), a, r, (s', i, \tau^{\text{exp}}))$
  Optimize θ and ψ with DDQN update with tuple $((s, i, \mu), a, r, (s', i, \mu))$
  Optimize ψ on $\nabla_\psi \min(\|f_\psi(\mu)\|_2^2, K)$ (Equation 3)
  Optimize ω on $\nabla_\omega \sum_t \|f_\psi(\mu) - g_\omega(\tau^{\text{exp}}_{:t})\|_2^2$ (Equation 4)

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

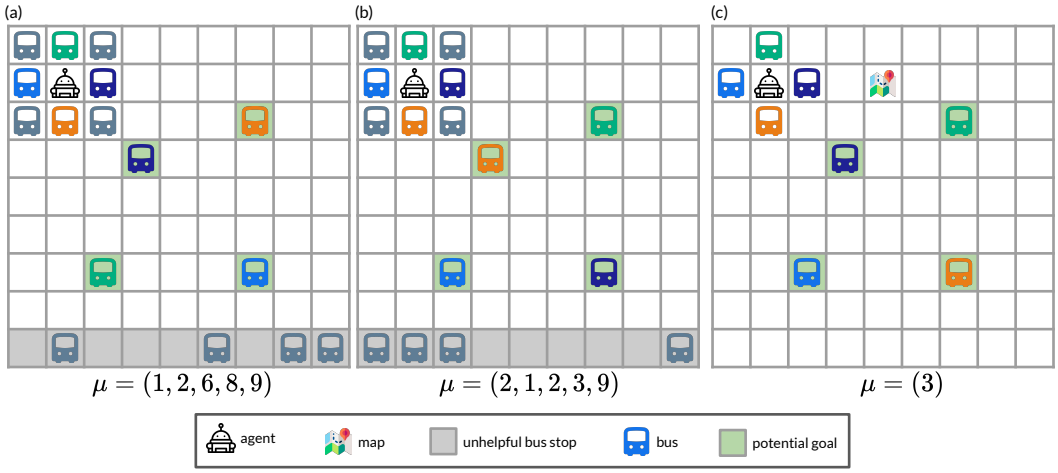

Figure 9: Examples of different *distracting bus* and *map* problems. (a) An example distracting bus problem. Though all unhelpful distracting buses are drawn in the same color (gray), the destinations of the gray buses are fixed within a problem: The 2nd coordinate of the problem specifies the destination (2nd cell from the left on the bottom row) of the gray bus to the upper-left of the agent, the 3rd coordinate of the problem specifies the destination (6th cell from the left on the bottom row) of the gray bus to the upper-right of the agent, and so on. (b) Another example distracting bus problem. The destinations of the helpful colored buses are a different permutation of the green potential goal locations, and the unhelpful gray buses lead to different destinations in the bottom row. (c) An example map problem. Touching the map reveals the destinations of the colored buses.

bus positions and their destinations are fixed within each problem. Additionally, in the distracting bus domain, the problem also encodes the x-coordinates of the destinations of the gray buses in the bottom gray row. Overall, in the map domain, the problem μ is a one-hot representation 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 one-hot value μ when the agent is at the map. In the distracting bus domain, the problem is represented as a 5-tuple, where the first index is the same as the map domain, and the remaining four indices describe the x-coordinates of the gray buses. Figure 9 displays three such examples.

Cooking. In different problems, the (color-coded) fridges contain 1 of 7 different ingredients. The ingredients in each fridge are unknown until the goes to the fridge and uses the pickup action. Figure 10 displays three example problems. The representation of the problem μ is a triple, where the i-th index is an indicator of what ingredient is in the i-th fridge.

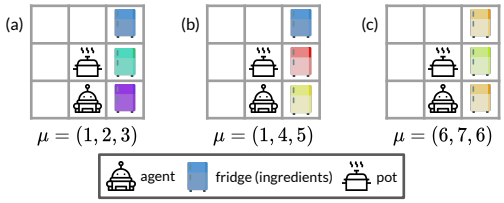

Figure 10: Three example cooking problems. The contents of the fridges (color-coded) are different in different problems.

Each instruction i corresponds to a recipe of placing the two correct ingredients in the pot in the right order. We represent this as a tuple, where the first index is an indicator of the first ingredient and the second index is the indicator of the second ingredient. In a given problem, we only sample recipes (instructions) involving the ingredients actually present in that problem. For test time, we hold out a randomly chosen 1% of the 7^3 different problems and 3 of the 7^2 different instructions: $(3, 6)$, $(7, 1)$, and $(5, 4)$.

We use a shaped 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 instruction; (ii) placing the first ingredient in the pot; (iii) picking up the second ingredient specified by the instruction; 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.

Standard meta-RL setting evaluation. In the standard meta-RL version of the cooking problems, the problem identifier is a 5-tuple, where the first 3 coordinates correspond to the contents of the fridges (the original problem identifier), and the last 2 coordinates specify the recipe (the instructions). 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). We do not hold out any problems for test time.

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 two problems $\mu = 0$ (the sign says "blue") and $\mu = 1$ (the sign says "red"). There are three instructions $i = 0, 1, 2$ corrsponding to picking up the ball, key, and 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 instruction) 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.

B.2 Additional Results

Distracting bus / map. Figure 11 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 9a). 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 9b). 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 execution, which requires more timesteps and therefore receives lower returns.

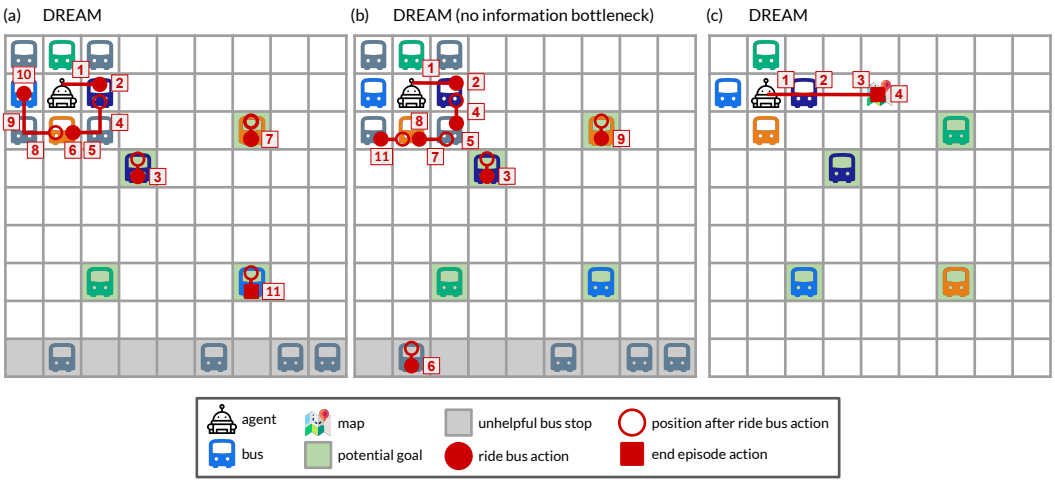

Figure 11: Examples of DREAM's learned exploration behavior. (a) DREAM learns the optimal exploration behavior on the *distraction* variant: riding 3 of the 4 helpful colored buses, which allows it to infer the destinations of all colored buses and efficiently solve any instruction during execution episodes. (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 execution 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.

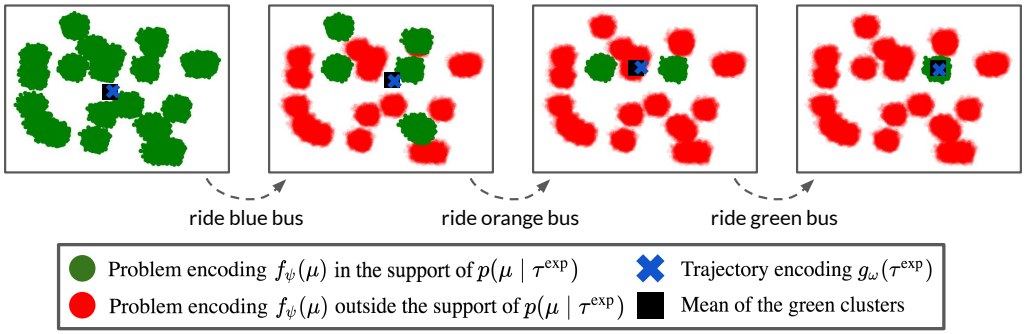

Figure 12: DREAM's learned encodings of the exploration trajectory and problems visualized with t-SNE [36].

In Figure 12, 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 [36]. 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 $(3, 3)$ when it has already been observed in τ^{exp} that riding the orange bus goes to location $(3, 3)$) 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 4.

Cooking. Figure 13 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 execution, which requires more timesteps and therefore receives lower returns.

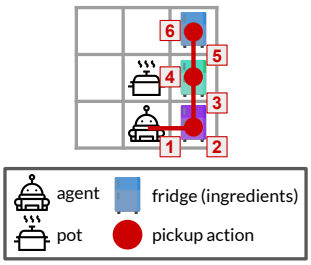

Figure 13: DREAM learns the optimal exploration policy, which learns the fridges' contents by going to each fridge and using the pickup action.

Sparse-reward 3D visual navigation. DREAM optimally explores by walking around the barrier and reading the sign. See `https://ezliu.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 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, RL^2, IMPORT, and VARIBAD.

State, instruction, and problem ID embeddings. All approaches learn embeddings of the state, instruction, 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, instruction, and problem ID embeddings below use this scheme.

Experience embeddings. 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 execution 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 execution. 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.
- For RL^2, IMPORT, VARIBAD, the same policy is used for both exploration and execution episodes. Therefore, they must also embed the current instruction i, as it is part of the state during execution episodes. During exploration episodes, we set i to be a special "no-instruction" instruction. These instructions are embedded as $e(i)$ according to the scheme described above.

Then, we apply a final linear layer with output dimension 64 to the concatenation of the above $[e(s_t); e(a_t); e(r_t); d_t]$ and including $e(i)$ for RL^2, IMPORT, and VARIBAD. 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 execution policy Q-values $\hat{Q}_\theta^{\text{exec}}(a \mid s, i, z)$, 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 and instructions with learned embedding functions $e(s)$ and $e_\theta(i)$ respectively. Then we apply a linear layer of output dimension 64 to the concatenation of $[e(s); e_\theta(i); z]$ 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.

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

IMPORT IMPORT also learns a recurrent policy $\pi(a_t \mid s_t, i, z)$, 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 $\hat{Q}(s_t, i, z, a_t)$. 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. 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. [18], 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 $\pi(a_t \mid i, z)$, 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 $\hat{Q}(s_t, i, z, a_t)$.

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 and instruction i as $e(s_t)$ and $e(i)$ and apply a linear layer of output dimension 64 to the concatenation $[e(s_t); e(i); z]$. 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

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.

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. [44] 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. [23]. and shared the same hyperparameter values across all methods for fairness. We optimize all methods with the Adam optimizer [20]. 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 execution 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. [19] 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 execution 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 $\lambda = 1$.

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 $V^*(i; \mu)$ be the maximum expected returns achievable by any execution policy with access to μ on instruction i and problem μ, i.e., with complete information. Let $\pi_\star^{exec}, \pi_\star^{exp}, F_\star$ and $q_\star(z \mid \tau^{exp})$ be the optimizers of the* DREAM *objective. Then for large enough T (the length of the exploration episode) and expressive enough function classes,*

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

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

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

where $V^{\pi^{\text{exec}}_\star}(i, z; \mu)$ is the expected returns of $\pi^{\text{exec}}_\star$ on problem μ given instruction i and 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,

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

as desired. We show that this occurs as follows.

Given stochastic encoder $F_\star(z \mid \mu)$, exploration policy $\pi^{\text{exp}}_\star$ maximizes $I(\tau^{\text{exp}}; z) = H(z) - H(z \mid \tau^{\text{exp}})$ (Equation 4) by assumption. Since only $H(z \mid \tau^{\text{exp}})$ depends on $\pi^{\text{exp}}_\star$, the exploration policy outputs trajectories that minimize

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

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 4, 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.

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 $\tau^{\text{exp}}_\star$ that reveals the problem μ; all other action sequences $\mathbf{a}$ lead to a trajectory $\tau^{\text{exp}}_{\mathbf{a}}$ that reveals no information. Similarly, the problem μ identifies a unique sequence of H actions $\mathbf{a}_\mu$ that receives reward 1 during execution, 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 execution.

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 execution and exploration episode is horizon H.
- The action space $\mathcal{A}$ consists of A discrete actions $\{1, 2, \ldots, A\}$.
- The space of problems $\mathcal{M} = \{1, 2, \ldots, |\mathcal{A}|^H\}$ 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 $(t, a_0, a_1, \ldots, a_{t-1}, 0, \ldots, 0, 0)$.
- 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 $s_{H-1} = (H - 1, a_0 = 1, \ldots, a_{H-2} = 1, 0, 0)$ (i.e., taking the entire action sequence $\mathbf{a}_\star$) transitions to the state $s_H = (H, a_0 = 1, \ldots, a_{H-2} = 1, a_{H-1} = 1, \mu)$, revealing the problem μ.

[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.

- 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 $(a_0, a_1, \ldots, a_{H-1})$ with $\sum_{t=0}^{H-1} a_t |\mathcal{A}|^t = \mu$. In problem μ with $\mathbf{a}_\mu = (a_0, a_1, \ldots, a_{H-1})$, taking action a_{H-1} at timestep $H-1$ at state $s_{H-1} = (H-1, a_0, a_1, \ldots, a_{H-2}, 0, 0)$ (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 $\tau^{\text{exp}}_{\mathbf{a}} = (s_0, a_0, r_0, \ldots, s_H)$ is uniquely identified by the action sequence $\mathbf{a}$ and the problem μ if revealed in s_H. We therefore write $\tau^{\text{exp}}_{\mathbf{a}} = (\mathbf{a}, \mu)$ for when $\mathbf{a} = \mathbf{a}_\star$ reveals the problem μ, and $\tau^{\text{exp}}_{\mathbf{a}} = (\mathbf{a}, 0)$, 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 $\hat{Q}^{\text{exec}}(\tau^{\text{exp}}, \mathbf{a})$ 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 execution Q-values $\hat{Q}^{\text{exec}}$. We estimate the execution Q-values $\hat{Q}^{\text{exec}}(\tau^{\text{exp}}, \mathbf{a})$ 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 $\tau^{\text{exp}}_{\mathbf{a}}$ is deterministically $(\mathbf{a}, 0)$, so we set $\hat{Q}^{\text{exp}}(\mathbf{a}) = \hat{V}^{\text{exec}}(\tau^{\text{exp}}_{\mathbf{a}}) = \max_{\mathbf{a}'} \hat{Q}^{\text{exec}}(\tau^{\text{exp}}_{\mathbf{a}}, \mathbf{a}')$. For $\mathbf{a}_\star$, the resulting exploration trajectory $\tau^{\text{exp}}_{\mathbf{a}_\star}$ 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 $\hat{V}^{\text{exec}}(\tau^{\text{exp}}_{\mathbf{a}_\star})$ of observed $\tau^{\text{exp}}_{\mathbf{a}_\star}$.

Recall that DREAM learns exploration Q-values $\hat{Q}^{\text{exp}}$ by regressing toward the learned decoder $\log \hat{q}(\mu \mid \tau^{\text{exp}}_{\mathbf{a}})$. We estimate the decoder $\hat{q}(\mu \mid \tau^{\text{exp}}_{\mathbf{a}})$ as the empirical counts of $(\mu, \tau^{\text{exp}}_{\mathbf{a}})$ divided by the empirical counts of $\tau^{\text{exp}}_{\mathbf{a}}$ and similarly estimate the Q-values as the empirical mean of $\log \hat{q}(\mu \mid \tau^{\text{exp}}_{\mathbf{a}})$. We denote the exploration Q-values learned after t timesteps as $\hat{Q}^{\text{exp}}_t$, 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 execution 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 execution 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 execution Q-values, the exploration Q-values are also slow to learn. In contrast, learning the decoder $\hat{q}(\mu \mid \tau^{\text{exp}}_{\mathbf{a}})$ is much faster, as it is independent of the execution 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., $\arg\max_{\mathbf{a}} \hat{Q}^{exp}_t(\mathbf{a}) = \mathbf{a}_\star$) 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, $\hat{Q}^{\text{exp}}_T(\mathbf{a}_\star) > \hat{Q}^{\text{exp}}_T(\mathbf{a})$ 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 $t \geq T$, $\hat{Q}^{\text{exp}}_t$ 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 $(\mu, \tau^{\text{exp}}_{\mathbf{a}})$ for two distinct μ for each action sequence $\mathbf{a}$ by casting this as a coupon collector problem, where each pair $(\mu, \tau^{\text{exp}}_{\mathbf{a}})$ 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 instruction values $\hat{V}_T^{\text{exec}}(\tau^{\text{exp}} = (\mathbf{a}_\star, \mu)) = 1$ for all μ in $\mathcal{M}$. Otherwise, there is a small, but non-zero probability that $\hat{V}_t^{\text{exec}}(\tau^{\text{exp}} = (\mathbf{a}, 0))$ will be greater at some $t > T$. For the instruction values to be optimal at all optimal exploration trajectories $\hat{V}_T^{\text{exec}}(\tau^{\text{exp}} = (\mathbf{a}_\star, \mu)) = 1$ 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 lowerbound the expecected 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 $\Omega(|\mathcal{A}|^{2H} \cdot |\mathcal{M}| \cdot \log |\mathcal{A}|^H)$. $\square$