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

Evan Zheran Liu [1] **Aditi Raghunathan** [1] **Percy Liang** [1] **Chelsea Finn** [1]

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 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. 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) (Duan et al., 2016; Wang et al., 2016a; Stadie et al., 2018; Zintgraf et al., 2019; Humplik et al., 2019; Kamienny et al., 2020) to maximize test returns on each meta-training environment and task, such that it can quickly adapt to new meta-test settings. 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). However, optimizing for exploration and execution end-to-end in this way creates a challenging chicken-and-egg optimization problem: 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

[1]Department of Computer Science, Stanford University, Stanford, USA. Correspondence to: Evan Z. Liu <evanliu@cs.stanford.edu>.

Proceedings of the 4th Lifelong Learning Workshop, Vienna, Austria, 2020. Copyright 2020 by the author(s).

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. We first train an execution policy to solve the tasks conditioned on an intermediate representation of this problem ID, and simultaneously minimize the mutual information between this ID representation and the ID itself. This makes the ID representation capture only task-relevant information, stripping extraneous information. 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 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 (Finn et al., 2017)) is unrealistic and can create unnecessary exploration challenges particularly in sparse reward settings. For example, the robot might 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). Providing instructions that contain information about the reward function also enables *reward-free adaptation*: the agent receives rewards during meta-training, but during meta-testing, it executes instructions in a new environment without ever receiving additional rewards, by exploring to identify relevant environment information and inferring the objective from the instructions (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) 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 rewards at meta-test time (Section 6).

2. Preliminaries

Meta-Reinforcement Learning Setting. The standard meta-RL setting considers a family of Markov decision processes (MDPs), $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}_\mathcal{T}, T_\mathcal{T} \rangle$ with states $\mathcal{S}$, actions $\mathcal{A}$, rewards $\mathcal{R}_\mathcal{T}$, and dynamics $T_\mathcal{T}$, parametrized by a *problem* $\mathcal{T}$. 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 $\mathcal{T} \sim p(\mathcal{T})$. Following prior problem settings (Finn et al., 2017; Rakelly et al., 2019; Rothfuss et al., 2018; Fakoor et al., 2019) and terminology from Duan et al. (2016), we define a *trial* as $M + N$ episodes in the same MDP $\mathcal{T}$, with M initial training (exploration) episodes of T steps (we set $M = 1$ for simplicity), 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 $\mathcal{T} \sim p(\mathcal{T})$ 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 (Rakelly et al., 2019; Humplik et al., 2019; Kamienny et al., 2020), we make the problem ID of $\mathcal{T}$ 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^{exe} used in execution episodes, but these policies may be the same or share parameters. The execution policy π^{exe} 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} in the exploration episode. The goal of the agent to maximize:

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

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

Learning to Reinforcement Learn. A common meta-RL approach is the *Learning to Reinforcement Learn* framework (Wang et al., 2016a; Duan et al., 2016; Zintgraf et al., 2019; Kamienny et al., 2020; Humplik et al., 2019), which directly maximizes the objective $\mathcal{J}$ in (1) by learning the same recurrent policy $\pi(a_t \mid s_t, \tau_{:t})$ for both exploration and execution (i.e., $\pi^{\text{exe}} = \pi^{\text{exp}} = \pi$). 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., $p(\mathcal{T} \mid \tau_{:t})$ (Kaelbling et al., 1998; Zintgraf et al., 2019).

The optimal policy explores to reduce this uncertainty in the initial exploration episode, ideally until the agent can receive high rewards via the same actions on all potential problems the agent might be in. Then, when uncertainty is low during the execution episodes, the optimal policy can take actions leading to high returns. 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 (Duan et al., 2016).

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. 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. On each episode, we sample an instruction i from a distribution of instructions $p_{\mathcal{T}}(i)$ and the agent receives the rewards $\mathcal{R}_{\mathcal{T}}(s_t, i)$ at each timestep t in problem $\mathcal{T}$. 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 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 (transition observations) and following the recipe on the fridge, without observing 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 receives no rewards at all. 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.

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 behavior on both exploration and execution episodes end-to-end by maximizing $\mathcal{J}$ in (1), can be sample inefficient in IMRL, even when $|\mathcal{I}| = 1$. For clarity, we refer to the policy followed during exploration episodes as π^{exp} and the policy followed during execution episodes as π^{exe}, although for RL^2 they are the same policy.

Figure 2a illustrates this end-to-end training approach. Note that π^{exe} 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^{exe}. If π^{exe} cannot effectively solve the task, then these gradients will be uninformative. This causes a bad local optimum: if our current (suboptimal) π^{exe} 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^{exe} from learning. Typically, early in training, both π^{exp} and π^{exe} 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 (Zhou et al., 2019b). 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 $F_\psi(z \mid \mathcal{T})$, which extracts only the information necessary to execute instructions in the problem $\mathcal{T}$. DREAM obtains this encoder by training an execution policy π^{exe} 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.

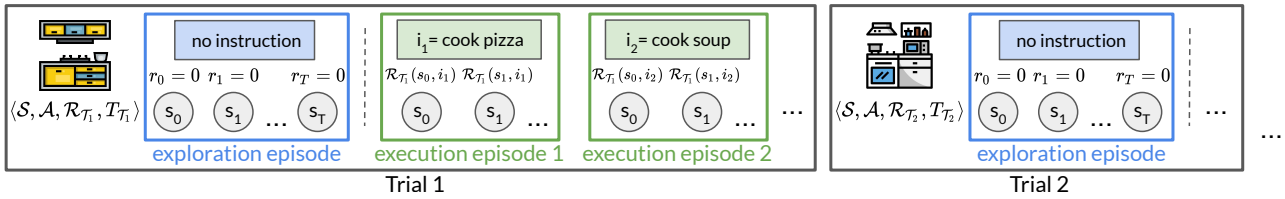

Figure 1. Reward-free instruction-based meta-RL: Meta-training trials (in a single problem) consist of a reward-free exploration episode followed by many execution episodes with rewards defined by instructions.

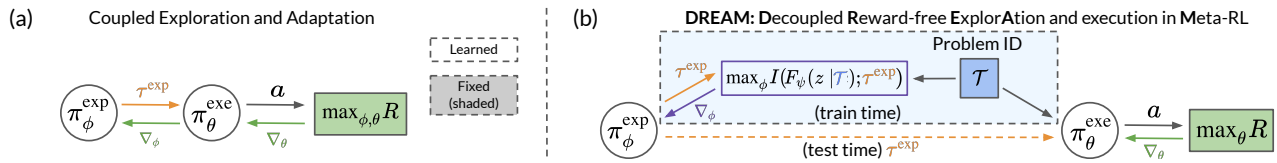

Figure 2. (a) Coupling between the exploration policy π^{exp} and execution policy π^{exe}. The exploration and execution policies are illustrated separately for clarity, but may be a single policy. Since the two policies depend on each other (for gradient signal and the τ^{exp} distribution), it is challenging to learn one when the other policy has not learned. (b) DREAM: π^{exp} and π^{exe} 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.

In this approach, the execution policy π^{exe} no longer relies on effective exploration π^{exp} to learn, and once $F_\psi(z \mid \mathcal{T})$ is learned, the the exploration policy π^{exp} can also learn independent of π^{exe}, decoupling the two optimization processes. Later, during meta-testing, when the problem ID $\mathcal{T}$ is unavailable, the two policies easily combine, since the trajectories generated by π^{exp} contain the same information as the stochastic encodings $F_\psi(z \mid \mathcal{T})$ that the execution policy π^{exe} trained on (overview in Figure 2b).

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

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

where $V^{\pi^{\text{exe}}_\theta}(i, z; \mathcal{T})$ is the expected return of π^{exe}_θ on problem $\mathcal{T}$ given instruction i and encoding z. Importantly, both terms are independent of the exploration policy π^{exp}.

We minimize the mutual information $I(z; \mathcal{T})$ by applying a variational upper bound (Barber & Agakov, 2003) as follows.

$$\begin{aligned} I(z; \mathcal{T}) &= \mathbb{E}_{\mathcal{T}} \left[\mathrm{D}_{\mathrm{KL}}(F_\psi(z \mid \mathcal{T}) || r(z)) \right] - \mathrm{D}_{\mathrm{KL}}(p_\psi(z) || r(z)) \\ &\leq \mathbb{E}_{\mathcal{T}} \left[\mathrm{D}_{\mathrm{KL}}(F_\psi(z \mid \mathcal{T}) || r(z)) \right], \end{aligned} \tag{3}$$

where r is any prior and z is distributed as $p_\psi(z) = \int_{\mathcal{T}} F_\psi(z \mid \mathcal{T}) p(\mathcal{T}) d\mathcal{T}$.

Learning an exploration policy given problem ID encodings. Once we've obtained an encoder $F_\psi(z \mid \mathcal{T})$ 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 (Barber & Agakov, 2003) as follows.

$$\begin{aligned} I(\tau^{\text{exp}}; z) &= H(z) - H(z \mid \tau^{\text{exp}}) \\ &\geq H(z) + \mathbb{E}_{\mathcal{T}, z\sim F_\psi, \tau^{\text{exp}}\sim\pi^{\text{exp}}} \left[\log q_\omega(z \mid \tau^{\text{exp}}) \right] \\ &= H(z) + \mathbb{E}_{\mathcal{T}, z\sim F_\psi} \left[\log q_\omega(z) \right] \\ &+ \mathbb{E}_{\mathcal{T}, z\sim F_\psi, \tau^{\text{exp}}\sim\pi^{\text{exp}}} \left[\sum_{t=1}^{T} \log q_\omega(z \mid \tau^{\text{exp}}_{:t}) - \log q_\omega(z \mid \tau^{\text{exp}}_{:t-1}) \right], \end{aligned} \tag{4}$$

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 $\mathcal{T}$ at test time and hence cannot sample from $F(z \mid \mathcal{T})$. 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:

$$r_t^{\text{exp}}(a_t, s_{t+1}, \tau_{t-1}^{\text{exp}}; \mathcal{T}) = \mathbb{E}_{z\sim F_\psi(z|\mathcal{T})}\left[\log q_\omega(z \mid [s_{t+1}; a_t; \tau_{:t-1}^{\text{exp}}]) - \log q_\omega(z \mid \tau_{:t-1}^{\text{exp}})\right] - c. \quad (5)$$

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 $z \sim F_\psi(z \mid \mathcal{T})$) than was already present in the trajectory $\tau_{:t-1}^{\text{exp}}$. The reward also includes a small per timestep penalty c to encourage exploring efficiently.

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 $\pi_\phi^{\text{exp}}(a_t \mid s_t, \tau_{:t}^{\text{exp}})$ parametrized by ϕ that conditions on its history $\tau_{:t}^{\text{exp}} = (s_0, a_0, r_0, \ldots, s_{t-1}, a_{t-1}, r_{t-1})$. 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 $F_\psi(z \mid \mathcal{T})$: For simplicity, we parameterize the stochastic encoder by learning a deterministic encoding $f_\psi(\mathcal{T})$ and apply Gaussian noise, i.e., $F_\psi(z \mid \mathcal{T}) = \mathcal{N}(f_\psi(\mathcal{T}), \rho^2 I)$. 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 $\|f_\psi(\mathcal{T})\|_2^2$.

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 $\mathbb{E}_{\mathcal{T}, z\sim F_\psi(z|\mathcal{T})}\left[\|z - g_\omega(\tau^{\text{exp}})\|_2^2\right]$ w.r.t., ω (Equation 4), and the exploration rewards take the form $r^{\text{exp}}(a, s', \tau^{\text{exp}}; \mathcal{T}) = \|f_\psi(\mathcal{T}) - g_\omega([\tau^{\text{exp}}; a; s'])\|_2^2 - \|f_\psi(\mathcal{T}) - g_\omega([\tau^{\text{exp}}])\|_2^2 - c$ (Equation 5).

3. Execution policy π_θ^{exe} and 4. Exploration policy π_ϕ^{exp}: We represent both policies as Q-networks and apply double deep Q-learning (DDQN) (van Hasselt et al., 2016), treating (s, i, z) as the state for π_θ^{exe}.

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_θ^{exe} are fixed. We also train π_θ^{exe} conditioned on the exploration trajectory $g_\omega(\tau^{\text{exp}})$, instead of exclusively training π^{exe} on the outputs of the encoder $z \sim F_\psi(z \mid \mathcal{T})$. 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 (Zhou et al., 2019b; Rakelly et al., 2019; Gupta et al., 2018; Gurumurthy et al., 2019), which also decouple exploration from execution, but do not recover the optimal policy even with infinite data. Formally,

Proposition 1. *Let $V^*(i; \mathcal{T})$ be the maximum expected returns achievable by any execution policy with access to $\mathcal{T}$ on instruction i and problem $\mathcal{T}$, i.e., with complete information. Let $\pi_\star^{exe}, \pi_\star^{exp}, F_\star$ 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 M (number of exploration episodes), and expressive enough function classes,*

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

Since knowledge of $\mathcal{T}$ uniquely identifies the MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}_{\mathcal{T}}, T_{\mathcal{T}} \rangle$, the maximum returns *any* agent can obtain corresponds to the returns of $\pi_{\text{opt}}^{\text{exe}}$, which has access to $\mathcal{T}$. Optimizing the decoupled objective in DREAM also obtains this maximum reward, which we prove in Appendix C.1.

5.2. Analysis of the Sample Complexity of Coupled and Decoupled Approaches

With enough capacity, approaches like RL^2 can also recover 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 $\mathcal{T}$; all other actions a reveal no information and lead to τ_a^{exp}. The problem $\mathcal{T}$ 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 $\Omega(\frac{|\mathcal{A}|^H}{H \log |\mathcal{A}|})$ 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: $\hat{Q}^{\text{exp}}(a) \leftarrow \hat{V}^{\text{exe}}(\tau_a^{\text{exp}}) := \max_{a'} \hat{Q}^{\text{exe}}(\tau_a^{\text{exp}}, a')$. We drop the fixed initial state from notation. This creates the local optimum in Section 4.1. Initially $\hat{V}^{\text{exe}}(\tau_\star^{\text{exp}})$ 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 $\hat{V}^{\text{exe}}(\tau_\star^{\text{exp}})$ 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 as (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 $\hat{q}$: $\hat{Q}^{\text{exp}}(a) \leftarrow \log \hat{q}(\mathcal{T} \mid \tau^{\text{exp}}(a))$. This decoder learns much faster than $\hat{Q}^{\text{exe}}$, 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

We compare DREAM with 3 state-of-the-art meta-RL algorithms using recurrent policies: (i) RL^2. (ii) VARIBAD (Zintgraf et al., 2019), which tries to optimally trades-off between exploration and exploitation with an auxiliary loss that predicts the dynamics and rewards from the recurrent state. (iii) IMPORT (Kamienny et al., 2020), which alternates between training the policy conditioned on the problem ID and its recurrent state and adds an auxiliary loss to keep the recurrent state close to the problem ID embedding. We report the average returns achieved by each method in IMRL trials with one reward-free exploration and one execution episode, averaged over 3 seeds with 1-std error bars (full details in Appendix B).

6.1. Didactic Experiments

We first evaluate on grid world domains illustrated in Figures 4a and 4b. The state is the agent's (x, y)-position and a one-hot indicator of the object at the agent's position (none, bus, map, pot, or fridge). The actions are to move up, down, left, or right; ride bus (if the agent is at a bus, this teleports the agent to another bus of the same color); pickup ingredients at the fridge; and drop ingredients at the pot. Episodes consist of 20 timesteps and the agent receives a reward of -0.1 at each timestep until the instruction is executed (full details in Appendix B.1 and qualitative results in Appendix B.2).

Targeted exploration. We first test if these methods can minimally explore to find the necessary information with the family of problems in Figure 4a. There are 4 possible instructions (green goal locations). Reaching the correct goal yields +1 reward and ends the episode. The four colored buses each lead to a different green goal location when ridden; in different problems $\mathcal{T}$, their destinations are set to 1 of the $4!$ different permutations. In the *distracting bus* version, in different problems, the gray buses lead to different gray locations in the bottom row, which is unhelpful for solving the task. In the *map* version, there is a map that reveals the destinations of the colored buses when touched.

Figures 5 and 6 (left) summarize the results. DREAM learns to optimally explore and thus receives optimal reward in both versions: in *distracting bus*, it rides 3 of the 4 colored buses, and in *map*, it visits the map and ends the exploration episode. During execution episodes, DREAM immediately reaches the goal by riding the correct colored bus. In contrast, IMPORT and RL^2 rarely explore buses during exploration and then walk to the goal during execution, which achieves lower reward, indicative of the coupling problem (Section 4.1). VARIBAD learns slower, likely from learning the dynamics in its auxiliary loss, but eventually matches IMPORT and E-RL^2 in ~3x as many samples. In *distracting bus*, we also compare to a version of DREAM *without* the information bottleneck (does not minimize the mutual information $I(z; \mathcal{T})$ between problem IDs and the encoder $F_\psi(z \mid \mathcal{T})$). 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 "pickup" action, and then directly retrieves the correct ingredients during execution. RL^2 only sometimes explores the fridges during training, leading to lower returns. It overfits to the training recipes and fails to cook the testing recipes. Training on the problem ID actually hurts IMPORT compared to RL^2. IMPORT successfully cooks conditioned on the problem ID, but it fails on many recipes conditioned on its recurrent state and we observe the recurrent state and problem ID embeddings

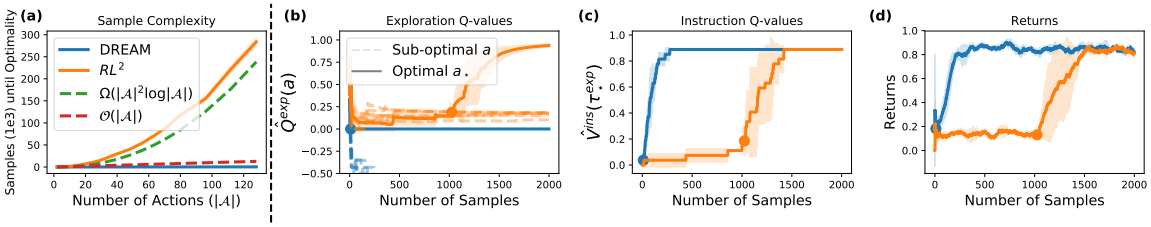

Figure 3. (a) Sample complexity of learning the optimal exploration policy as the action space $|\mathcal{A}|$ grows (1000 seeds). (b) Exploration Q-values $\hat{Q}^{\text{exp}}(a)$. The policy $\arg\max_a \hat{Q}^{\text{exp}}(a)$ is optimal after the dot; (c) instruction values given optimal trajectory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

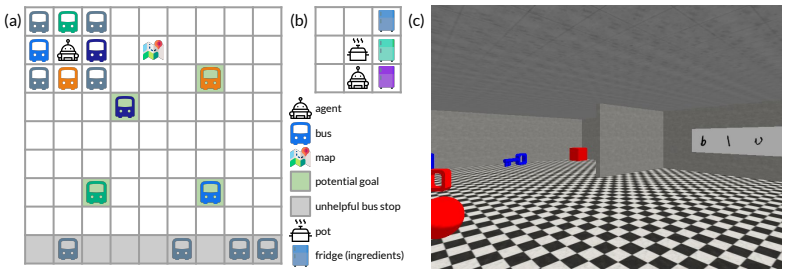

Figure 4. (a) Navigation. (b) Cooking. (c) 3D Visual Navigation: the agent must read the sign to determine what colored object to go to.

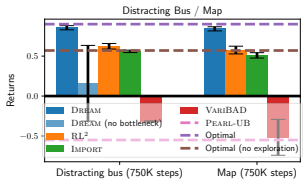

Figure 5. Navigation results. Only DREAM optimally explores all the buses and read the map.

become far apart. VARIBAD learns slowly as before.

6.2. Sparse-Reward 3D Visual Navigation

We conclude with a more challenging variant of the problem family introduced by Kamienny et al. (2020) with visual observations, illustrated in (Figure 4c): 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 6 (right)). 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 (Schmidhuber, 1987; Thrun & Pratt, 2012; Naik & Mammone, 1992; Bengio et al., 1991; 1992; Hochreiter et al., 2001; Andrychowicz et al., 2016; Santoro et al., 2016). 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 (Finn et al., 2017; Rothfuss et al., 2018; Agarwal et al., 2019; Yang et al., 2019; Mendonca et al., 2019), learning a model (Nagabandi et al., 2018; Sæmundsson et al., 2018), multi-task learning (Fakoor et al., 2019), or leveraging demonstrations (Zhou et al., 2019a). 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 (Duan et al., 2016; Wang et al., 2016a; Mishra et al., 2017; Stadie et al., 2018; Zintgraf et al., 2019; Humplik et al., 2019; Kamienny et al., 2020; Ritter et al., 2018). and can represent the optimal policy (Kaelbling et al., 1998), 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) (Thompson, 1933; Rakelly et al., 2019), obtaining exploration trajectories predictive of dynamics or rewards (Zhou et al., 2019b;

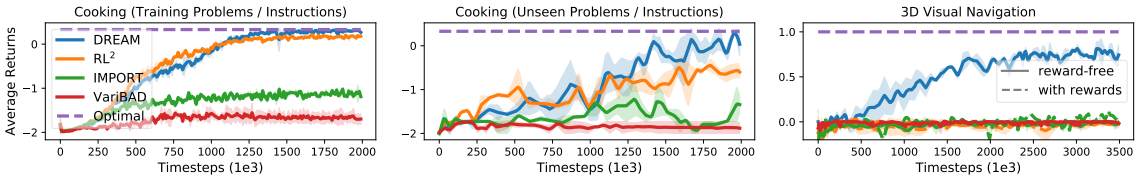

Figure 6. Left / Middle: Cooking results. Only DREAM achieves optimal reward on both unseen and training problems and instructions. Right: Visual navigation results. Only DREAM reads the sign and solves the task.

Gurumurthy et al., 2019), or exploration noise (Gupta et al., 2018). These avoid the problem of coupling in the first category, but no longer learn optimal exploration. In particular, TS (Rakelly et al., 2019) explores by guessing the task and executing a policy for that task, and therefore cannot represent exploration behaviors that are different from adaptation (Russo et al., 2017). Predicting the dynamics (Zhou et al., 2019b; Gurumurthy et al., 2019) 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 (Gregor et al., 2016; Houthooft et al., 2016; Eysenbach et al., 2018; Warde-Farley et al., 2018) 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 (Stadie et al., 2018) and when rewards are unavailable at test time (Yang et al., 2019). Other work (Nagabandi et al., 2018; Sæmundsson et al., 2018) assume that the reward function is known to the agent, like instructions.

8. Conclusion

In summary, we present two contributions. First, we identify a coupling issue with existing meta-RL works that leads to poor sample complexity, and propose decoupled objectives for exploration and execution (DREAM) to address this. Unlike prior decoupled objectives (Zhou et al., 2019b; Rakelly et al., 2019; Gupta et al., 2018; Gurumurthy et al., 2019) that do not learn the optimal exploration strategy, maximizing our decoupled objectives also maximizes returns. Second, we propose a new meta-RL setting (IMRL) to model realistic situations where the agent receives instructions. The combination of these two contributions is greater than their sum: In IMRL, DREAM adapts to new environments and tasks without ever seeing rewards during meta-testing, solving challenging problems that prior methods fail to solve. IMRL also alleviates unnecessary exploration for inferring the task. We therefore advocate for research on adaptation to provide the agent instructions like in IMRL. While our work provides many benefits, we study the setting where the agent is allowed initial training 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.

Acknowledgements

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 2018255189. Icons used in this paper were made by Freepik, ThoseIcons and dDara from www.flaticon.com.

References

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

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

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

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

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

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

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. RL^2: Fast reinforcement learn-

ing via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Stadie, B., Yang, G., Houthooft, R., Chen, P., Duan, Y., Wu, Y., Abbeel, P., and Sutskever, I. The importance of

sampling inmeta-reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 9280–9290, 2018.

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

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

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

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

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

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

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

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

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

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

Zintgraf, L., Shiarlis, K., Igl, M., Schulze, S., Gal, Y., Hofmann, K., and Whiteson, S. 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 (Wang et al., 2016b; van Hasselt et al., 2016), 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 $\hat{Q}^{\text{exe}}(s, i, z, a; \theta)$ 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 replay buffer (lines 10-13). We train the execution policy on both stochastic encodings of the problem ID $\mathcal{N}(f_\psi(\mathcal{T}), \rho^2 I)$ 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 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 take the following gradient step, regressing toward the target network:

$$\nabla_\phi \left\| \hat{Q}^{\text{exp}}(s_t, \tau^{\text{exp}}_{:t-1}, a_t; \phi) - \text{target} \right\|_2^2$$
$$\text{where target} = r_t^{\text{exp}} + \gamma \hat{Q}^{\text{exp}}(s_{t+1}, [\tau^{\text{exp}}_{:t-1}; a_t; s_t], a_{\text{DDQN}}; \phi'),$$
$$\text{and } a_{\text{DDQN}} = \arg\max_a \hat{Q}^{\text{exp}}(s_{t+1}, [\tau^{\text{exp}}_{:t-1}; a_t; s_t]; \phi).$$

We perform a similar update with the execution Q-value parameters (lines 17-19). We sample 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. These take the following form:

$$\nabla_{\theta,\omega} \left\| \hat{Q}^{\text{exe}}(s, i, g_\omega(\tau^{\text{exp}}), a; \theta) - \text{target}_{\text{traj}} \right\|_2^2$$
$$\text{and } \nabla_{\theta,\psi} \left\| \hat{Q}^{\text{exe}}(s, i, f_\psi(\mathcal{T}), a; \theta) - \text{target}_{\text{prob}} \right\|_2^2,$$
$$\text{where target}_{\text{traj}} = r + \hat{Q}^{\text{exe}}(s', i, g_{\omega'}(\tau^{\text{exp}}), a_{\text{traj}}; \theta')$$
$$a_{\text{traj}} = \arg\max_a \hat{Q}^{\text{exe}}(s', i, g_\omega(\tau^{\text{exp}}), a; \theta),$$
$$\text{and target}_{\text{prob}} = r + \hat{Q}^{\text{exe}}(s', i, f_{\psi'}(\mathcal{T}), a_{\text{prob}}; \theta')$$
$$a_{\text{prob}} = \arg\max_a \hat{Q}^{\text{exe}}(s', i, f_\psi(\mathcal{T}), a; \theta).$$

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). Since the magnitude $\|f_\psi(\mathcal{T})\|_2^2$ partially determines the scale of the reward, we add a hyperparameter K and only minimize the magnitude when it is larger than K. 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. Tasks

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. 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 $\mathcal{T}$ 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 $\mathcal{T}$ 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 7 displays three such examples. This figure also shows the optimal exploration policies learned by DREAM. When we remove the information bottleneck from DREAM, as in Figure 5, the exploration begins to visit the gray buses, because these are part of the problem representation, even though this results in lower execution returns.

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 8 displays three example problems and the optimal exploration learned by DREAM. The representation of the problem $\mathcal{T}$ is a triple, where the i-th index is an indicator of what ingredient is in the i-th fridge.

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 $\mathcal{A}_\mathcal{T}$. 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.

Sparse-reward 3D visual navigation. We implement this domain in Gym MiniWorld (Chevalier-Boisvert, 2018), where the agent's observations are $80 \times 60 \times 3$ RGB arrays. There are two problems $\mathcal{T} = 0$ (the sign says "blue") and $\mathcal{T} = 1$ (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 $\mathcal{A}_\mathcal{T}(s, i)$ is +1 for picking up the correct colored object (according to $\mathcal{T}$) 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. Figures 7a, 7b, and 7c show 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 7a). 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 7b). 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.

In Figure 9, we additionally visualize the exploration trajectory encodings $g_\omega(\tau^{\text{exp}})$ and problem ID encodings $f_\psi(\mathcal{T})$ that DREAM learns in the distracting bus domain by applying t-SNE (van der Maaten & Hinton, 2008). 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 $p(\mathcal{T} \mid \tau^{\text{exp}})$ are drawn in green, while problems outside the support (e.g., a problem that specifies that riding the green bus goes to location $(3, 3)$

Algorithm 1 DREAM DDQN

Initialize execution replay buffer $\mathcal{B}_{\text{exe}} = \{\}$ and exploration replay buffer $\mathcal{B}_{\text{exp}} = \{\}$
Initialize execution Q-value $\hat{Q}^{\text{exe}}$ 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 $\mathcal{T} \sim p(\mathcal{T})$, defining MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}_\mathcal{T}, T_\mathcal{T} \rangle$
  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 $\mathcal{B}_{\text{exp}} = \mathcal{B}_{\text{exp}} \cup \{(s_t, a_t, s_{t+1}, \mathcal{T}, \tau^{\text{exp}})\}_t$.

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

  Sample batches of $(s_t, a_t, s_{t+1}, \mathcal{T}, \tau^{\text{exp}}) \sim \mathcal{B}_{\text{exp}}$ from exploration replay buffer.
  Compute reward $r^{\text{exp}}_t = \|f_\psi(\mathcal{T}) - g_\omega(\tau^{\text{exp}}_{:t})\|_2^2 - \left\|f_\psi(\mathcal{T}) - g_\omega(\tau^{\text{exp}}_{:t-1})\right\|_2^2 - c$ (Equation 5).
  Optimize ϕ with DDQN update with tuple $(s_t, a_t, r^{\text{exp}}_t, s_{t+1})$

  Sample batches of $(s, a, r, s', i, \mathcal{T}, \tau^{\text{exp}}) \sim \mathcal{B}_{\text{exe}}$ 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 $((s, i, \mathcal{T}), a, r, (s', i, \mathcal{T}))$
  Optimize ψ on $\nabla_\psi \min(\|f_\psi(\mathcal{T})\|_2^2, K)$ (Equation 3)
  Optimize ω on $\nabla_\omega \sum_t \|f_\psi(\mathcal{T}) - g_\omega(\tau^{\text{exp}}_{:t})\|_2^2$ (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

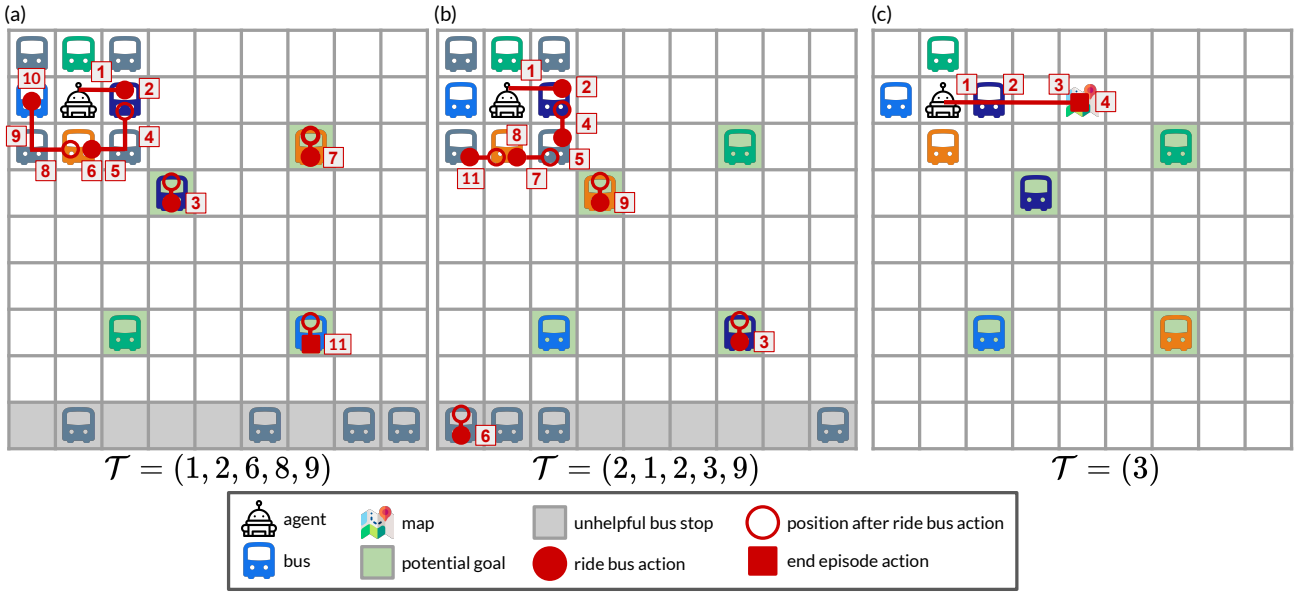

Figure 7. Examples of the *distracting bus* and *map* problems. (a) Example of the exploration policy DREAM learns in distracting bus. It infers the destinations of all colored buses by riding 3 of the 4 colored buses. (b) Example of the exploration policy DREAM (no bottleneck) learns in distracting bus. It rides the gray buses (whose destinations are fixed throughout a trial) because they are part of the problem, but this does not help it achieve high returns. (c) Example of the exploration DREAM learns in map. It learns the destinations of all colored buses by visiting the map.

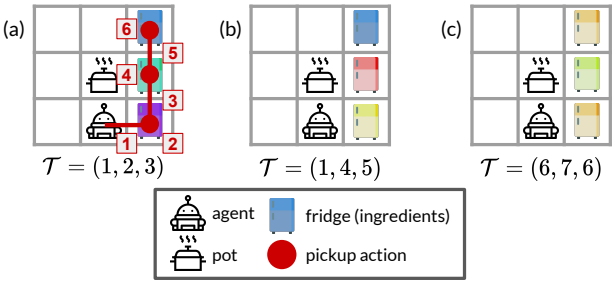

Figure 8. Example of cooking problems. (a) Example of the exploration policy DREAM learns. It learns the fridges' contents by going to each fridge and using the pickup action.

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 8a 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.

B.3. Model Architecture

In this section, we describe the model architectures used in our experiments. Where possible, we use the same model architecture for all methods: DREAM, RL^2, IMPORT, and VARIBAD.

DREAM For the decoder $g_\omega(\tau^{\text{exp}} = (s_0, a_0, s_1, \ldots, s_T))$, we embed each transition (s_t, a_t, s_{t+1}) of the exploration trajectory τ^{exp} as follows. We apply a state learned embedding function to both states $e_\omega(s_t)$ and $e_\omega(s_{t+1})$ and embed the action $e_\omega(a_t)$ by applying a learned embedding matrix with output dimension 32. Then we apply two linear layers with output dimension 128 and 64 respectively, using ReLU activations to obtain the embedding for each transition. 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 $\hat{Q}^{\text{exe}}_\theta(a \mid s, i, z)$, we either choose z to be the decoder embedding of the exploration trajectory $g_\omega(\tau^{\text{exp}})$ of to be an embedding of the problem ID $e_\theta(\mathcal{T})$, 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 $\hat{Q}^{\text{exp}}_\phi(a_t \mid s_t, \tau^{\text{exp}}_{:t})$, we embed the past states, actions, and rewards recurrently with an LSTM of output dimension 64. We embed each state, action, and reward with learned embedding functions $e_\phi(s)$, $e_\phi(a)$, $e_\phi(r)$ respectively, where the actions are embedded with a learned embedding matrix of output dimension 16 and the rewards are embedded with a linear layer with output dimension 16. Then, we pass a linear layer with output dimension 64 and ReLU activations over their concatenation $[e_\phi(s_t); e_\phi(a_t); e_\phi(r_t)]$ at each timestep t. We then pass an LSTM with output dimension 64 over this, setting the initial hidden state to all 0. 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 $\tau_{:t}$. We parametrize this dueling double Q-networks, learning Q-values $\hat{Q}(s_t, i, \tau_{:t}, a_t)$. We use an identical architecture to DREAM's exploration policy Q-values, except we also embed the instructions and episode terminations. Concretely, we learn embedding functions on the state $e(s)$, action $e(a)$, rewards $e(r)$, episode terminations $e(d)$, and instructions $e(i)$ and pass a linear layer of output dimension 64 over the concatenation of $[e(s_t); e(a_t); e(r_t); e(d_t); e(i_t)]$ with a ReLU activation at each timestep t. Then, we pass an LSTM with output dimension 64 over this. We embed actions and episode terminations with a learned embedding matrix with output dimension 16 and embed rewards with a learned linear layer with output dimension 16. As for all policies, we apply two final linear layer heads to obtain value and advantage estimates.

IMPORT IMPORT learns a policy $\pi(a_t \mid s_t, i, z)$ producing actions a_t given the state s_t, instructions i and embedding z of either the problem $\mathcal{T}$ or the history $\tau_{:t}$ We also parametrize this dueling double Q-networks, learning Q-values $\hat{Q}(s_t, i, z, a_t)$, using a similar architecture to RL^2.

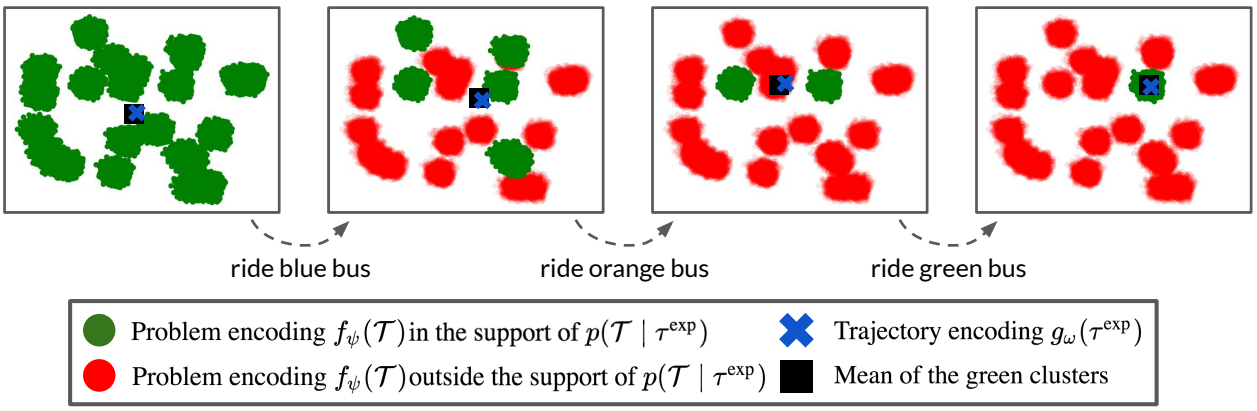

Figure 9. DREAM's learned encodings of the exploration trajectory and problems visualized with t-SNE (van der Maaten & Hinton, 2008).

We embed the history $\tau_{:t}$ and instruction i to obtain $e(\tau_{:t})$ identically as RL^2 above. We embed the problem $\mathcal{T}$ and current state and instructions (s_t, i) with learned embedding functions $e(\mathcal{T})$, $e(s_t)$, and $e(i)$ respectively. Then we alternate meta-training trials between choosing $z = e(\mathcal{T})$ and $z = e(\tau_{:t})$. We apply a linear layer of output dimension 64 to the concatenation $[e(s_t); e(i); z]$ with ReLU activations and then apply two linear layer heads to obtain value and advantage estimates.

VARIBAD VARIBAD also 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 $\tau_{:t}$. We also parametrize this dueling double Q-networks, learning Q-values $\hat{Q}(s_t, i, z, a_t)$, using a similar architecture to above. To embed the history and instructions, VARIBAD learns an encoder $\text{enc}(z \mid \tau_{:t})$, which we parametrize by embedding the history $\tau_{:t}$ using the same architecture as RL^2 to obtain $e(\tau_{:t})$. Then we apply a linear layer with output dimension to obtain a mean μ. Finally, we sample z from $\mathcal{N}(\mu, \nu^2 I)$, where $\nu^2 = 0.00001$. We also tried learning the variance instead of fixing it to $\nu^2 I$ by applying a second linear head to the $e(\tau_{:t})$, but found no change in performance, so stuck with the simpler fixed variance approach. Given z sampled from the encoder, we embed the current state s_t with a learned encoder $e(s_t)$ 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 also learns a state decoder $\hat{T}(s' \mid a, s, z)$ and reward decoder $\hat{\mathcal{R}}(s' \mid a, s, z)$ for its auxiliary losses, where z is sampled from the decoder. For both of these, we embed the action $e(a)$ with an embedding matrix of output dimension 32 and learn a state embedder $e(s)$. Then we apply two linear layers with output dimension 128 to the concatenation $[e(s); e(a); z]$. Finally, we apply two linear heads, one for the state decoder and one for the reward decoder and take the mean-squared error. 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. (2019) 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.

State, instruction, and problem ID embeddings. For all learned embeddings of the state, instruction and problem ID embeddings, we embed each dimension independently with an embedding matrix of dimension 32. Then we concatenate the per-dimension embeddings and apply two linear layers with output dimension 256 and 64 with ReLU activations.

In the 3D visual navigation task, we use the architecture recommended by Chevalier-Boisvert (2018) for the learned state embeddings instead of the embedding scheme described above. 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 in between the 3 CNN layers and apply a final linear layer to the flattened output of output dimension 128.

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. (2015). and shared the same hyperparameter values across all methods for fairness. We optimize all methods with the Adam optimizer (Kingma & Ba, 2014). Table 1 summarizes the shared hyperparam-

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.

eters 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. (2019) 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. *Let $V^*(i; \mathcal{T})$ be the maximum expected returns achievable by any execution policy with access to $\mathcal{T}$ on instruction i and problem $\mathcal{T}$, i.e., with complete information. Let $\pi_\star^{exe}, \pi_\star^{exp}, F_\star$ 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 M (number of exploration episodes), and expressive enough function classes,*

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

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

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

where $V^{\pi_\star^{\text{exe}}}(i, z; \mathcal{T})$ is the expected returns of $\pi_\star^{\text{exe}}$ on problem $\mathcal{T}$ 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 $F_\star(z \mid \mathcal{T})$ for all exploration trajectories in the support of $\pi^{\text{exp}}(\tau^{\text{exp}} \mid \mathcal{T})$[1], for each problem $\mathcal{T}$. Then,

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

as desired. We show that this occurs as follows.

Given stochastic encoder $F_\star(z \mid \mathcal{T})$, exploration policy $\pi_\star^{\text{exp}}$ 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 $\pi_\star^{\text{exp}}$, the exploration policy outputs trajectories that minimize

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

where $p(z \mid \tau^{\text{exp}})$ is the true conditional distribution and $H(F_\star(z \mid \mathcal{T}), p(z \mid \tau^{\text{exp}}))$ is the cross-entropy. The cross-entropy is minimized when $p(z \mid \tau^{\text{exp}}) = F_\star(z \mid \mathcal{T})$, which can be achieved with enough exploration episodes M and long enough exploration trajectories T (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 $F_\star(z \mid \mathcal{T})$ 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

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

before) in the exploration episode leads to a trajectory $\tau_\star^{\text{exp}}$ that reveals the problem $\mathcal{T}$; all other action sequences $\mathbf{a}$ lead to a trajectory $\tau_{\mathbf{a}}^{\text{exp}}$ that reveals no information. Similarly, the problem $\mathcal{T}$ identifies a unique sequence of H actions $\mathbf{a}_\mathcal{T}$ 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 $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}_\mathcal{T}, T_\mathcal{T} \rangle$ parametrized by the problem ID $\mathcal{T} \in \mathcal{M}$, where:

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

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

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

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

In this particular tabular example with deterministic dynamics that encode the entire action history and rewards, learning a per timestep Q-value is equivalent to learning a Q-value for the entire trajectory. Hence, we denote exploration Q-values $\hat{Q}^{\text{exp}}(\mathbf{a})$ estimating the returns from taking the entire sequence of H actions $\mathbf{a}$ at the initial state s_0 and exeuction Q-values $\hat{Q}^{\text{exe}}(\tau^{\text{exp}}, \mathbf{a})$ 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 $\hat{Q}^{\text{exe}}$. We estimate the execution Q-values $\hat{Q}^{\text{exe}}(\tau^{\text{exp}}, \mathbf{a})$ as the sample mean of returns from taking actions $\mathbf{a}$ given the exploration trajectory τ^{exp} and estimate the exploration Q-values $\hat{Q}^{\text{exp}}(\mathbf{a})$ as the sample mean of the targets. More precisely, for action sequences $\mathbf{a} \neq \mathbf{a}_\star$, the resulting exploration trajectory $\tau_{\mathbf{a}}^{\text{exp}}$ is deterministically $(\mathbf{a}, 0)$, so we set $\hat{Q}^{\text{exp}}(\mathbf{a}) = \hat{V}^{\text{exe}}(\tau_{\mathbf{a}}^{\text{exp}}) = \max_{\mathbf{a}'} \hat{Q}^{\text{exe}}(\tau_{\mathbf{a}}^{\text{exp}}, \mathbf{a}')$. For $\mathbf{a}_\star$, the resulting exploration trajectory $\tau_{\mathbf{a}_\star}^{\text{exp}}$ may be any of $(\mathbf{a}_\star, \mathcal{T})$ for $\mathcal{T} \in \mathcal{M}$, so we set $\hat{Q}^{\text{exp}}(\mathbf{a}_\star)$ as the empirical mean of $\hat{V}^{\text{exe}}(\tau_{\mathbf{a}_\star}^{\text{exp}})$ of observed $\tau_{\mathbf{a}_\star}^{\text{exp}}$.

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

Given this setup, we are ready to state the formal sample complexity results. Intuitively, learning the execution Q-values for RL^2 is slow, because, in problem $\mathcal{T}$, it involves observing the optimal exploration trajectory from taking actions $\mathbf{a}_\star$ and then observing the corresponding execution actions $\mathbf{a}_\mathcal{T}$, 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 $\hat{q}(\mathcal{T} \mid \tau_{\mathbf{a}}^{\text{exp}})$ 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 1. *Let T be the number of samples from uniform random exploration such that the greedy-exploration policy is guaranteed to be optimal (i.e., $\arg\max_{\mathbf{a}} \hat{Q}_t^{exp}(\mathbf{a}) = \mathbf{a}_\star$) for all $t \geq T$. If $\hat{Q}^{exp}$ is learned with* DREAM*, the expected value of T is $\mathcal{O}(|\mathcal{A}|^H \log |\mathcal{A}|^H)$. If $\hat{Q}^{exp}$ is learned with RL^2, the expected value of T is $\Omega(|\mathcal{A}|^{2H} |\mathcal{M}| \log |\mathcal{A}|^H)$.*

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

For RL^2, the exploration policy is optimal for all timesteps $t \geq T$ for some T only if the instruction values $\hat{V}_T^{\text{exe}}(\tau^{\text{exp}} = (\mathbf{a}_\star, \mathcal{T})) = 1$ for all $\mathcal{T}$ in $\mathcal{M}$. Otherwise, there is a small, but non-zero probability that $\hat{V}_t^{\text{exe}}(\tau^{\text{exp}} = (\mathbf{a}, 0))$ will be greater at some $t > T$. For the instruction values to be optimal at all optimal exploration trajectories $\hat{V}_T^{\text{exe}}(\tau^{\text{exp}} = (\mathbf{a}_\star, \mathcal{T})) = 1$ for all $\mathcal{T} \in \mathcal{M}$, we must jointly observe exploration trajectory $\tau^{\text{exp}} = (\mathbf{a}_\star, \mathcal{T})$ and corresponding action sequence $\mathbf{a}_{\mathcal{T}}$ for each problem $\mathcal{T} \in \mathcal{M}$. We can lowerbound the expecected number of samples required to observe this by casting this as a coupon collector problem, where each pair $(\tau^{\text{exp}} = (\mathbf{a}_\star, \mathcal{T}), \mathbf{a}_{\mathcal{T}})$ 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 $\Omega(|\mathcal{A}|^{2H} \cdot |\mathcal{M}| \cdot \log |\mathcal{A}|^H)$. □