A. DREAM Training Details

Algorithm 2 summarizes a practical algorithm for training DREAM. We parametrize both the exploration and exploitation policies as recurrent 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 exploitation Q-values $\hat{Q}^{\text{task}}(s, z, a; \theta)$ parametrized by θ (and target network parameters θ'). We train on trials with one exploration and one exploitation episode, but can test on arbitrarily many exploitation episodes, as the exploitation policy acts on each episode independently (i.e. it does not maintain a hidden state across episodes). Using the choices for F_ψ and q_ω in Section 4.3, with target parameters ψ' and ω' respectively, training proceeds as follows.

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

Next, we sample from the replay buffers and update the parameters. First, we sample $(s_t, a_t, s_{t+1}, \mu, \tau^{\text{exp}})$-tuples from the exploration replay buffer and perform a normal DDQN update on the exploration Q-value parameters ϕ using rewards computed from the decoder (lines 13-15). Concretely, we minimize the following standard DDQN loss function w.r.t., the parameters ϕ, where the rewards are computed according to Equation 5:

$$\mathcal{L}_{\text{exp}}(\phi) = \mathbb{E}\left[\left\|\hat{Q}^{\text{exp}}(s_t, \tau^{\text{exp}}_{:t-1}, a_t; \phi) - \text{target}\right\|_2^2\right],$$

$$\begin{aligned}
\text{where target} &= (r^{\text{exp}}_t + \gamma \hat{Q}^{\text{exp}}(s_{t+1}, [\tau^{\text{exp}}_{:t-1}; a_t; s_t], a_{\text{DDQN}}; \phi'),\\
r^{\text{exp}}_t &= \left\|f_\psi(\mu) - g_\omega(\tau^{\text{exp}}_{:t})\right\|_2^2 - \\
&\quad \left\|f_\psi(\mu) - g_\omega(\tau^{\text{exp}}_{:t-1})\right\|_2^2 - c,\\
a_{\text{DDQN}} &= \arg\max_a \hat{Q}^{\text{exp}}(s_{t+1}, [\tau^{\text{exp}}_{:t-1}; a_t; s_t]; \phi).
\end{aligned}$$

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

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

$$\begin{aligned}
\text{where target}_{\text{id}} &= (r + \hat{Q}^{\text{task}}(s', f_{\psi'}(\mu), a_{\text{id}}; \theta'),\\
a_{\text{id}} &= \arg\max_a \hat{Q}^{\text{task}}(s', f_\psi(\mu), a; \theta).
\end{aligned}$$

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

$$\begin{aligned}
\text{where target}_{\text{traj}} &= (r + \hat{Q}^{\text{task}}(s', g_{\omega'}(\tau^{\text{exp}}), a_{\text{traj}}; \theta'),\\
a_{\text{traj}} &= \arg\max_a \hat{Q}^{\text{task}}(s', g_\omega(\tau^{\text{exp}}), a; \theta),
\end{aligned}$$

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

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

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

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

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

As is standard with deep Q-learning (Mnih et al., 2015), 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 20-22).

B. Experiment Details

B.1. Problem Details

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

In the map domain, the problem μ is an integer representing which of the $4!$ permutations of the four green goal loca-

Algorithm 2 DREAM DDQN

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

10:    Compute embedding $z \sim \mathcal{N}(f_\psi(\mu), \rho^2 I)$ on trial $\equiv 0 \pmod 2$ and $z = g_\omega(\tau^{\text{exp}})$ on trial $\equiv 1 \pmod 2$.
11:    Roll-out $\epsilon$-greedy exploitation policy $\hat{Q}^{\text{task}}(s_t, z, a_t; \theta)$, producing trajectory $(s_0, a_0, r_0, \dots)$ with $r_t = \mathcal{R}_\mu(s_{t+1})$.
12:    Add tuples to the exploitation replay buffer $\mathcal{B}_{\text{task}} = \mathcal{B}_{\text{task}} \cup \{(s_t, a_t, r_t, s_{t+1}, \mu, \tau^{\text{exp}})\}_t$.

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

16:    Sample batches of $(s, a, r, s', \mu, \tau^{\text{exp}}) \sim \mathcal{B}_{\text{task-id}}$ from exploitation replay buffer.
17:    Optimize $\theta$ and $\psi$ with DDQN update with tuple $((s, \mu), a, r, (s', \mu))$ with $\mathcal{L}_{\text{task-id}}(\theta, \psi)$ on trial $\equiv 0 \pmod 2$.
18:    Optimize $\theta$ and $\omega$ with DDQN update with tuple $((s, \tau^{\text{exp}}), a, r, (s', \tau^{\text{exp}}))$ with $\mathcal{L}_{\text{task-traj}}(\theta, \omega)$ on trial $\equiv 1 \pmod 2$.
19:    Optimize $\psi$ on $\mathcal{L}_{\text{bottleneck}}(\psi) = \nabla_\psi \min(\|f_\psi(\mu)\|_2^2, K)$
20:    Optimize $\omega$ on $\mathcal{L}_{\text{decoder}}(\omega) = \nabla_\omega \sum_t \|f_\psi(\mu) - g_\omega(\tau^{\text{exp}}_{:t})\|_2^2$ (Equation 4)

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

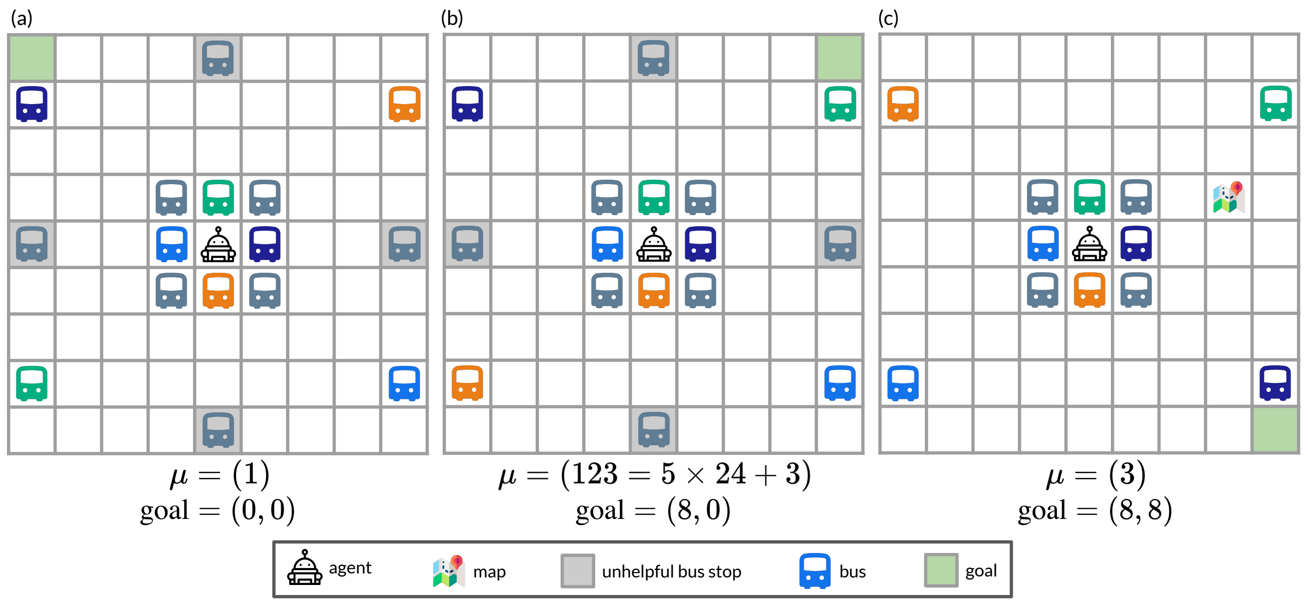

Figure 9. Examples of different *distracting bus* and *map* problems. (a) An example distracting bus problem. Though all unhelpful distracting buses are drawn in the same color (gray), the destinations of the gray buses are fixed within a problem. (b) Another example distracting bus problem. The destinations of the helpful colored buses are a different permutation (the orange and green buses have swapped locations). This takes on permutation $3 \equiv \mu \pmod{4!}$, instead of 1. The unhelpful gray buses are also a different permutation (not shown), taking on permutation $5 = \lfloor \frac{\mu}{4!} \rfloor$. (c) An example map problem. Touching the map reveals the destinations of the colored buses, by adding μ to the state observation.

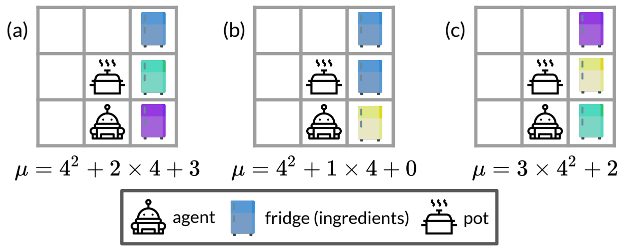

Figure 10. Three example cooking problems. The contents of the fridges (color-coded) are different in different problems.

tions the colored buses map to. The states include an extra dimension, which is set to 0 when the agent is not at the map, and is set to this integer value μ when the agent is at the map. Figure 9 displays three such examples.

Cooking. In different problems, the (color-coded) fridges contain 1 of 4 different ingredients. The ingredients in each fridge are unknown until the goes to the fridge and uses the pickup action. Figure 10 displays three example problems. The problem ID μ is an integer between 0 and 4^3, where $\mu = 4^2a + 4b + c$ indicates that the top right fridge has ingredient a, the middle fridge has ingredient b and the bottom right fridge has ingredient c.

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

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

Cooking without goals. While we evaluate on goal-conditioned benchmarks to deepen the exploration challenge, forcing the agent to discover all the relevant information for *any* potential goal, many standard benchmarks (Finn et al., 2017; Yu et al., 2019) don't involve goals. We therefore include a variant of the cooking task, where there are no goals. We simply concatenate the goal (recipe) to the problem ID μ. Additionally, we modify the rewards so that picking up the second ingredient yields $+0.25$ and dropping it yields -0.25 reward, so that it is possible to infer the recipe from the rewards. Finally, to make the problem harder, the agent cannot pick up new ingredients unless its inventory is empty (by using the drop action), and we also increase the number of ingredients to 7. The results are in Section B.2.

Sparse-reward 3D visual navigation. We implement this domain in Gym MiniWorld (Chevalier-Boisvert, 2018), where the agent's observations are $80 \times 60 \times 3$ RGB arrays. There are three problems $\mu = 0$ (the sign says "blue"), $\mu = 1$ (the sign says "red"), and $\mu = 2$ (the sign says "green"). There are two goals, represented as 0 and 1, corresponding to picking up the key and the box, respectively. The reward function $\mathcal{A}_\mu(s, i)$ is +1 for picking up the correct colored object (according to μ) and the correct type of object (according to the goal) and -1 for picking up an object of the incorrect color or type. Otherwise, the reward is 0. On each episode, the agent begins at a random location on the other side of the barrier from the sign.

B.2. Additional Results

Analysis of the learned policies. Please see `https://ezliu.github.io/dream/` for videos and analysis of the exploration and exploitation behavior learned by DREAM and other approaches, which is described in the text below.

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 exploitation, which requires more timesteps and therefore receives lower returns.

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 (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 $p(\mu \mid \tau^{\text{exp}})$ are drawn in green, while problems outside the support (e.g., a problem

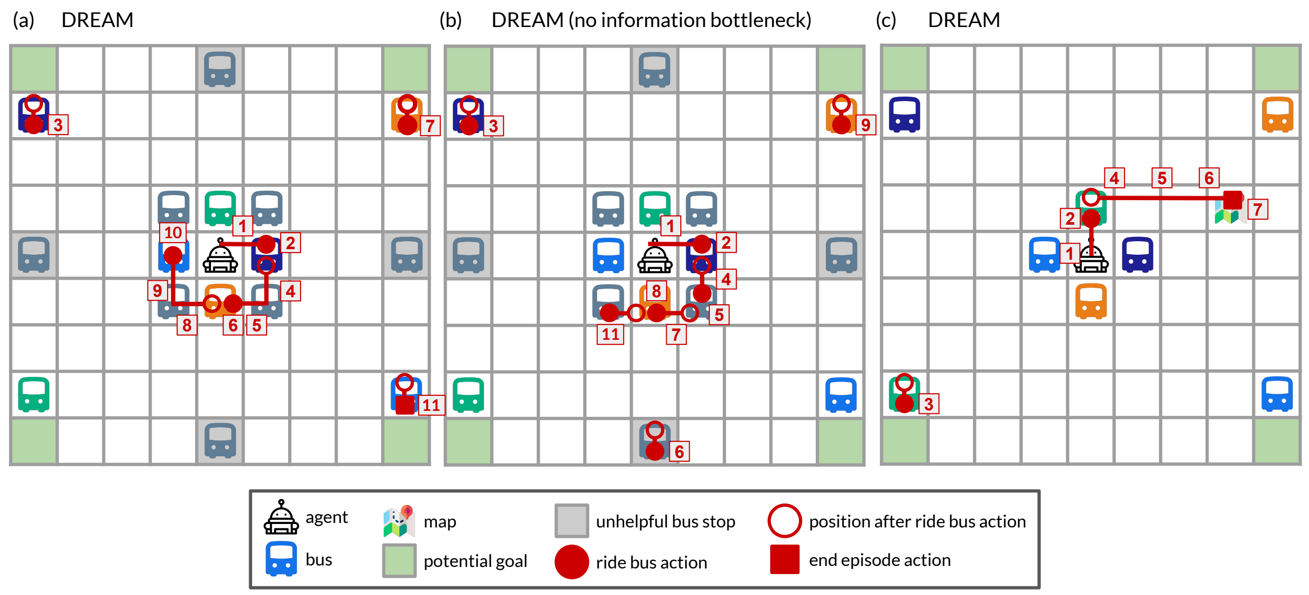

Figure 11. Examples of DREAM's learned exploration behavior. (a) DREAM learns the optimal exploration behavior on the *distraction* variant: riding 3 of the 4 helpful colored buses, which allows it to infer the destinations of all colored buses and efficiently reach any goal during exploitation episodes. (a) Without the information bottleneck, DREAM also explores the unhelpful gray buses, since they are part of the problem. This wastes exploration steps, and leads to lower returns during exploitation episodes. (c) DREAM learns the optimal exploration on the *map* variant: it goes to read the map revealing all the buses' destinations, and then ends the episode, though it unnecessarily rides one of the buses.

that specifies that riding the green bus goes to location $(0, 1)$ when it has already been observed in τ^{exp} that riding the orange bus goes to location $(0, 1)$) are drawn in red. We also plot the encoding of the exploration trajectory τ^{exp} so far as a blue cross and the mean of the green clusters as a black square. We find that the encoding of the exploration trajectory $g_\omega(\tau^{\text{exp}})$ tracks the mean of the green clusters until the end of the exploration episode, when only one cluster remains, and the destinations of all the colored buses has been discovered. Intuitively, this captures uncertainty in what the potential problem ID may be. More precisely, when the decoder is a Gaussian, placing $g_\omega(\tau^{\text{exp}})$ at the center of the encodings of problems in the support exactly minimizes Equation 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 exploitation, which requires more timesteps and therefore receives lower returns.

Cooking without goals. We provide additional results in the case where the cooking domain is modified to not include goals (see Section B.1). The results are summarized in Figure 14 and show the same trends as the results in original cooking benchmark. DREAM learns to optimally explore by investigating the fridges, and then also optimally exploits, by directly collecting the relevant ingredients. The next best approach E-RL^2, only sometimes explores the fridges, again getting stuck in a local optimum, yielding only slightly higher reward than no exploration at all.

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

Robustness to imperfections in the problem ID. Recall that the problem ID is a simple and easy-to-provide unique one-hot for each problem. We test of DREAM is robust to imperfections in the problem ID by assigning each problem in the *map* benchmark 3 different problem IDs. When a problem is sampled during meta-training, it is randomly labeled with 1 of these 3 different problem IDs. We find that this imperfection in the problem ID does not impact DREAM's final performance at all: it still achieves optimal exploitation returns of 0.8 after 1M time steps of training.

Robustness to hyperparameters. DREAM uses 3 additional hyperparameters on top of standard RL algorithm hyperparameters: the information bottleneck weight λ, en-

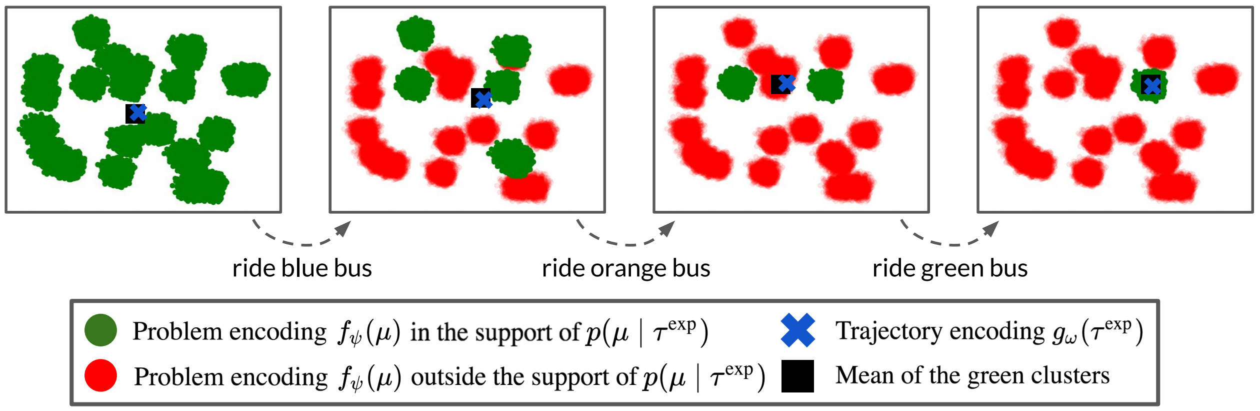

Figure 12. DREAM's learned encodings of the exploration trajectory and problems visualized with t-SNE (van der Maaten & Hinton, 2008).

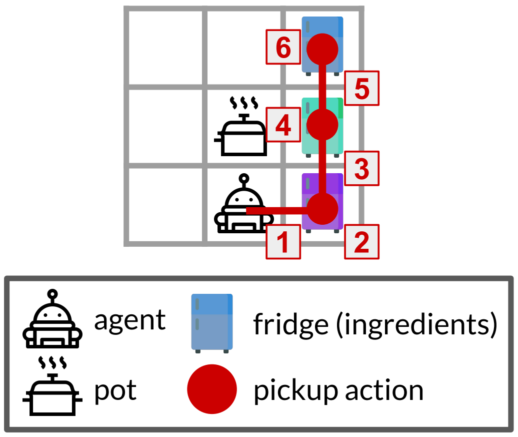

Figure 13. DREAM learns the optimal exploration policy, which learns the fridges' contents by going to each fridge and using the pickup action.

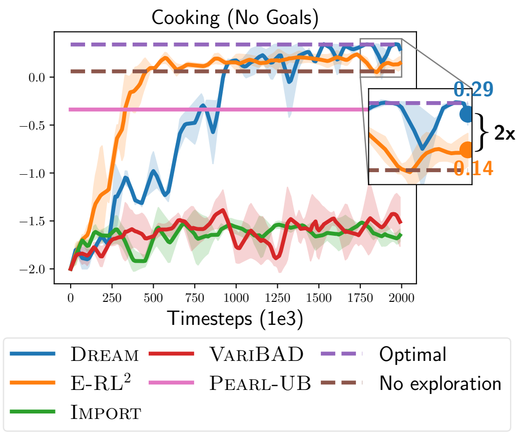

Figure 14. Cooking without goals results. Only DREAM learns the optimal policy, achieving ~2x more reward than the next best approach.

coder and decoder variance ρ, and per timestep exploration penalty c. We test how sensitive DREAM is to these hyperparameter values by evaluating DREAM with 3 different values for each of these hyperparameters, while holding the other hyperparameter values constant, set to the values described in Section B.4. Specifically, we evaluate the following values: $\lambda = [0.1, 1, 3], \rho = [0.01, 0.1, 0.3], c = [0, 0.01, 0.1]$. Across all four domains (*distracting bus*, *map*, *cooking*, and *3D visual navigation*), DREAM achieves near-optimal returns for all values except $\lambda = 3$, suggesting that DREAM is fairly robust to hyperparameter choices.

B.3. Other Approaches and Architecture Details

In this section, we detail the loss functions that E-RL^2, IMPORT, and VARIBAD optimize, as well as the model architectures used in our experiments. Where possible, we use the same model architecture for all methods: DREAM, E-RL^2, IMPORT, and VARIBAD. All approaches are implemented in PyTorch (Paszke et al., 2017), using a DQN implementation adapted from Liu et al. (2020b) and code adapted from Liu et al. (2020a).

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

In the 3D visual navigation task, we use a different embedding scheme for the states, as they are images. We apply 3 CNN layers, each with 32 output layers and stride length 2, and with kernel sizes of 5, 5, and 4 respectively. We apply

ReLU activations between the CNN layers and apply a final linear layer to the flattened output of the CNN layers, with an output dimension of 128.

All state and problem ID embeddings below use this scheme.

Experience embeddings. E-RL^2, IMPORT, VARIBAD and the exploration and exploitation policies in DREAM also learn an embedding of the history of prior experiences $\tau = (s_0, a_0, r_0, s_1, \dots)$ 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.
- Embed the rewards with a linear layer of output dimension 16. We set r_{-1} to be 0.
- Embed the episode termination d_t as $e(d_t)$ with an embedding matrix of output dimension 16. Note that d is true during all episode terminations within a trial for RL^2, IMPORT, and VARIBAD.

Then, we apply a final linear layer with output dimension 64 to the concatenation of the above $[e(s_t); e(a_t); e(r_t); d_t]$. Finally, to obtain an embedding of the entire history τ, 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 $g_\omega(\tau^{\text{exp}} = (s_0, a_0, r_0, s_1, \dots, s_T))$, 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}. Then, given embeddings for each transition, we embed the entire trajectory by passing an LSTM with output dimension 128 on top of the transition embeddings, followed by two linear layers of output dimension 128 and 64 with ReLU activations.

For the exploitation policy Q-values $\hat{Q}_\theta^{\text{task}}(a_t \mid s_t, \tau_{:t}, z)$, during meta-testing, we choose z to be the decoder embedding of the exploration trajectory $g_\omega(\tau^{\text{exp}})$, and during meta-training, we choose z to be the the embedding of the problem ID $e_\theta(\mu)$. To embed the history $\tau_{:t} = (s_0, a_0, r_0, \dots, s_{t-1}, a_{t-1}, r_{t-1})$, we use the experience embedding scheme described above to obtain an embedding $e(\tau_{:t})$, omitting the reward embeddings for simplicity. We embed the state with a learned embedding functions $e(s)$. Then we apply a linear layer of output dimension 64 to the concatenation of $[e(s_t); e(\tau_{:t}), z]$ with a ReLU activation. Finally, we apply two linear layer heads of output dimension 1 and $|\mathcal{A}|$ respectively to form estimates of the value and advantage functions, using the dueling Q-network parametrization. To obtain Q-values, we add the value function to the advantage function, subtracting the mean of the advantages.

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

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

IMPORT IMPORT also learns a recurrent policy $\pi(a_t \mid s_t, z)$, but conditions on the embedding z, which is either an embedding of the problem μ or the history $\tau_{:t}$. We also parametrize this policy with dueling double Q-networks, learning Q-values $\hat{Q}(s_t, z, a_t)$. We embed the state s_t as $e(s_t)$, the problem μ as $e_\phi(\mu)$ and the history $\tau_{:t}$ as $e_\theta(\tau_{:t})$ using the previously described embedding schemes. 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. (2020), we use $\beta = 1$ in our final experiments, and found that performance changed very little when we experimented with other values of β.

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

VARIBAD encodes the belief state with an encoder enc$(z \mid s_t, \tau\colon t)$. Our implementation of this encoder embeds s_t and $\tau_{:t}$ using the same experience embedding approach as above, and use the output as the mean m for a distribution. Then,

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.

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

VARIBAD does not update the encoder via gradients through the policy. Instead, VARIBAD jointly trains the encoder with state decoder $\hat{T}(s' \mid a, s, z)$ and reward decoder $\hat{\mathcal{R}}(s' \mid a, s, z)$, where z is sampled from the encoder, as follows. Both decoders embed the action a as $e(a)$ with an embedding matrix of output dimension 32 and embed the state s as $e(s)$. Then we apply two linear layers with output dimension 128 to the concatenation $[e(s); e(a); z]$. Finally, we apply two linear heads, one for the state decoder and one for the reward decoder and take the mean-squared error with the true next state s' and the true rewards r respectively. In the 3D visual navigation domain, we remove the state decoder, because the state is too high-dimensional to predict. Note that Zintgraf et al. (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.

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, 2015). Table 1 summarizes the shared hyperparameters used by all methods and we detail any differences in hyperparameters between the methods below.

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

We train the recurrent policies (DREAM's exploration and exploitation policies, 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, we additionally use per timestep exploration reward penalty $c = 0.01$, decoder and stochastic encoder variance $\rho^2 = 0.1$, and information bottleneck weight $\lambda = 1$. Note that this information bottleneck weight λ could be adapted via dual gradient descent to solve the constrained optimization problem in (2), but we find that dynamically adjusting λ is not necessary for good performance. For the MiniWorld experiments, we use $c = 0$.

C. Analysis

C.1. Consistency

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

Proposition 1. *Assume $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}_\mu, \mathcal{T}_\mu \rangle$ is ergodic for all problems $\mu \in \mathcal{M}$. Let $V^*(\mu)$ be the maximum expected returns achievable by any exploitation policy with access to the problem ID μ, i.e., with complete information. Let $\pi_\star^{task}, \pi_\star^{exp}, F_\star$ and $q_\star(z \mid \tau^{exp})$ be the optimizers of the* DREAM *objective. Then, if the function classes* DREAM *optimizes over are well-specified, there exists a finite T such that if the length of the exploration episode is at least T,*

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

Proof. Recall that $\pi_\star^{\text{task}}$ and $F_\star(z \mid \mu)$ are optimized to solve the constrained optimization in (2). In particular, they must satisfy the constraint, so $\pi_\star^{\text{task}}$ achieves the desired expected returns conditioned on the stochastic encoding of the problem $F_\star(z \mid \mu)$:

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

where $V^{\pi_\star^{\text{task}}}(z; \mu)$ is the expected returns of $\pi_\star^{\text{task}}$ on problem μ given embedding z. Therefore, it suffices to show that the distribution over z from the decoder $q_\star(z \mid \tau^{\text{exp}})$ is equal to the distribution from the encoder $F_\star(z \mid \mu)$ for all exploration trajectories in the support of $\pi^{\text{exp}}(\tau^{\text{exp}} \mid \mu)$[1], for

[1]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.

each problem μ. Then,

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

as desired. We show that this occurs as follows.

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

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

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. $\square$

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 $\tau_\star^{\text{exp}}$ that reveals the problem μ; all other action sequences $\mathbf{a}$ lead to a trajectory $\tau_{\mathbf{a}}^{\text{exp}}$ that reveals no information. Similarly, the problem μ identifies a unique sequence of H actions $\mathbf{a}_\mu$ that receives reward 1 during exploitation, while all other action sequences receive reward 0. Again, taking the action sequence $\mathbf{a}_\star$ during exploration is therefore necessary and sufficient to obtain optimal reward 1 during exploitation.

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

- Each exploitation and exploration episode is horizon H.
- The action space $\mathcal{A}$ consists of A discrete actions $\{1, 2, \ldots, A\}$.
- 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 $|\mathcal{A}|$ 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.
- 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 $\tau_{\mathbf{a}}^{\text{exp}} = (s_0, a_0, r_0, \ldots, s_H)$ is uniquely identified by the action sequence $\mathbf{a}$ and the problem μ if revealed in s_H. We therefore write $\tau_{\mathbf{a}}^{\text{exp}} = (\mathbf{a}, \mu)$ for when $\mathbf{a} = \mathbf{a}_\star$ reveals the problem μ, and $\tau_{\mathbf{a}}^{\text{exp}} = (\mathbf{a}, 0)$, otherwise.

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

In this particular tabular example with deterministic dynamics that encode the entire action history and rewards, learning a per timestep Q-value is equivalent to learning a Q-value for the entire trajectory. Hence, we denote exploration

Q-values $\hat{Q}^{\text{exp}}(\mathbf{a})$ estimating the returns from taking the entire sequence of H actions $\mathbf{a}$ at the initial state s_0 and exeuction Q-values $\hat{Q}^{\text{task}}(\tau^{\text{exp}}, \mathbf{a})$ estimating the returns from taking the entire sequence of H actions $\mathbf{a}$ at the initial state s_0 given the exploration trajectory τ^{exp}. We drop s_0 from notation, since it is fixed.

Recall that RL^2 learns exploration Q-values $\hat{Q}^{\text{exp}}$ by regressing toward the exploitation Q-values $\hat{Q}^{\text{task}}$. We estimate the exploitation Q-values $\hat{Q}^{\text{task}}(\tau^{\text{exp}}, \mathbf{a})$ as the sample mean of returns from taking actions $\mathbf{a}$ given the exploration trajectory τ^{exp} and estimate the exploration Q-values $\hat{Q}^{\text{exp}}(\mathbf{a})$ as the sample mean of the targets. More precisely, for action sequences $\mathbf{a} \neq \mathbf{a}_\star$, the resulting exploration trajectory $\tau^{\text{exp}}_{\mathbf{a}}$ is deterministically $(\mathbf{a}, 0)$, so we set $\hat{Q}^{\text{exp}}(\mathbf{a}) = \hat{V}^{\text{task}}(\tau^{\text{exp}}_{\mathbf{a}}) = \max_{\mathbf{a}'} \hat{Q}^{\text{task}}(\tau^{\text{exp}}_{\mathbf{a}}, \mathbf{a}')$. 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 $\hat{V}^{\text{task}}(\tau^{\text{exp}}_{\mathbf{a}_\star})$ 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 exploitation Q-values for RL^2 is slow, because, in problem μ, it involves observing the optimal exploration trajectory from taking actions $\mathbf{a}_\star$ and then observing the corresponding exploitation actions $\mathbf{a}_\mu$, which only jointly happens roughly once per $|\mathcal{A}|^{2H}$ samples. Since RL^2 regresses the exploration Q-values toward the exploitation Q-values, the exploration Q-values are also slow to learn. In contrast, learning the decoder $\hat{q}(\mu \mid \tau^{\text{exp}}_{\mathbf{a}})$ is much faster, as it is independent of the exploitation actions, and in particular, already learns the correct value from a single sample of $\mathbf{a}_\star$. We formalize this intuition in the following proposition, which shows that DREAM learns in a factor of at least $|\mathcal{A}|^H |\mathcal{M}|$ fewer samples than RL^2.

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