

A Experimental details

A.1 Environments

In both environments below, we transform 2D states and goals into 82×82 pixel observations.

Pixel continuous point maze: This continuous 2D maze environment is taken from [Trott et al., 2019]. The action space is a continuous vector $(\delta x, \delta y) = [0, 1]^2$. Original states and goals are 2D (x, y) positions in the maze and success is achieved if the L2 distance between states and goal coordinates is below $\delta = 0.15$ (only used during the evaluation of success coverage), while the overall size of the mazes is 6×6 . The episode rollout horizon is $T = 50$ steps. Examples of pixel observation goals are shown in Figure 5.

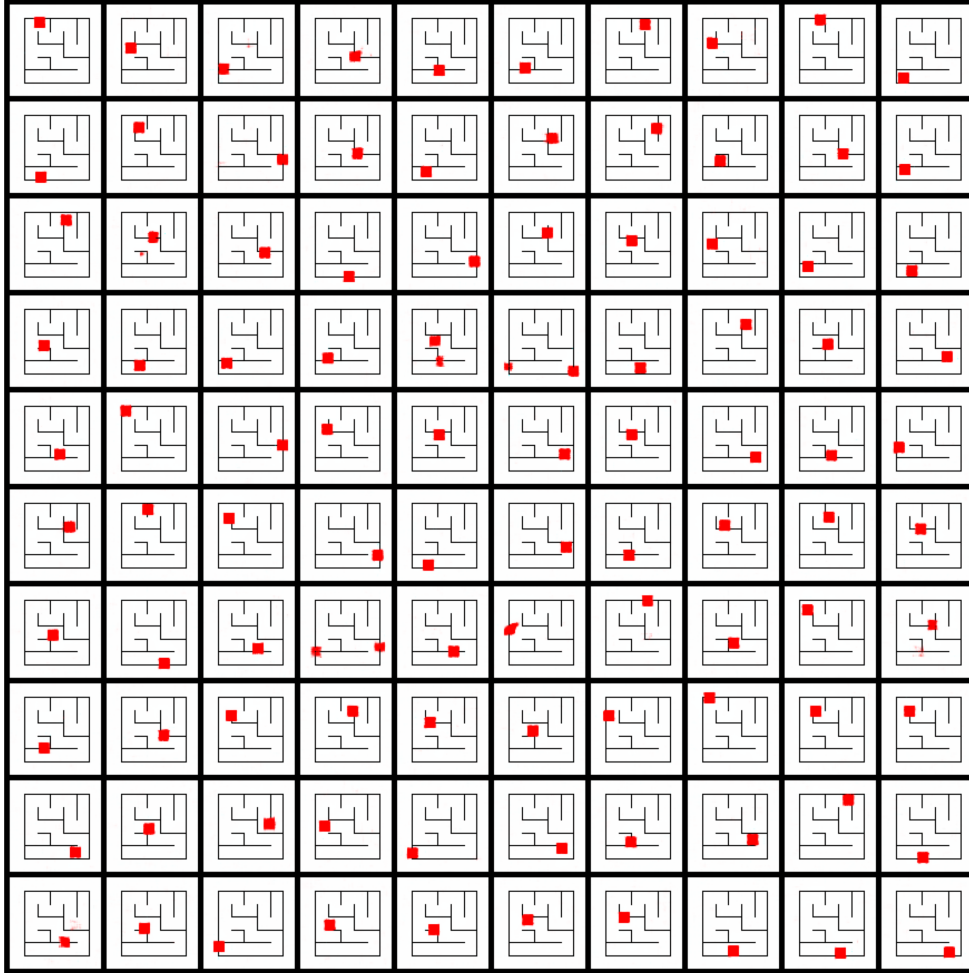


Figure 5: Example of decoded pixel goals in maze environment: we sample latent goals from the latent prior $z \sim p(z) = \mathcal{N}(0, I)$ and plot the corresponding decoded pixel goals $p_\theta(x|z)$. Images were obtained using the decoder trained with DRAG.

Pixel Reach hard walls: This environment is adapted from the Reach-v2 MetaWorld benchmark [Yu et al., 2021] where the gripper is initially stuck between four walls and has to navigate carefully between them to reach the goals. The original observations are 49-dimensional vectors containing the gripper position as well as other environment variables, the actions and the goals are 3-dimensional corresponding to (x, y, z) coordinates. Success is achieved if the L2 distance between states and goal

864 coordinates is less than $\delta = 0.1$ (only used during the evaluation of success coverage). The episode
 865 rollout horizon is $T = 300$ steps.

866 We transform states and goals into pixels observation using the Mujoco rendering function with the
 867 following camera configuration:

```
868 DEFAULT_CAMERA_CONFIG = {
869     "distance": 2.,
870     "azimuth": 270,
871     "elevation": -30.0,
872     "lookat": np.array([0, 0.5, 0]),
873 }
```

874 The walls configuration is obtained with the addition of the following bodies into the "worldbody" of
 875 the xml file of the original environment:

```
876 <body name="wall_1" pos="0.15 0.55 .2">
877     <geom material="wall_brick" type="box" size=".005 .24 .2" rgba="0 1 0 1"/>
878     <geom class="wall_col" type="box" size=".005 .24 .2" rgba="0 1 0 1"/>
879 </body>
880
881 <body name="wall_2" pos="-0.15 0.55 .2">
882     <geom material="wall_brick" type="box" size=".005 .24 .2" rgba="0 1 0 1"/>
883     <geom class="wall_col" type="box" size=".005 .24 .2" rgba="0 1 0 1"/>
884 </body>
885
886 <body name="wall_3" pos="0.0 0.65 .2">
887     <geom material="wall_brick" type="box" size=".4 .005 .2" rgba="0 1 0 1"/>
888     <geom class="wall_col" type="box" size=".4 .005 .2" rgba="0 1 0 1"/>
889 </body>
890
891 <body name="wall_4" pos="0.0 0.35 .2">
892     <geom material="wall_brick" type="box" size=".4 .005 .2" rgba="0 1 0 1"/>
893     <geom class="wall_col" type="box" size=".4 .005 .2" rgba="0 1 0 1"/>
894 </body>
```

895 Examples of pixel observation goals are shown in Figure 6.

896 A.2 β -VAE

897 A.2.1 Training schedule

898 During the first 300k steps of the agent, we train the VAE every 5k agent steps for 50 epochs of 10
 899 optimization steps (on a dataset of 1000 inputs uniformly sampled from the buffer, divided in 10
 900 minibatches of 100 examples). Afterward, we train it every 10k agent steps.

901 The following other schedules have been experimented, each getting worse average results for any
 902 algorithm:

- 903 1. During the first 300k steps of the agent, train the VAE every 10k agent steps. Afterward,
 904 train it every 20k steps.
- 905 2. During the first 100k steps of the agent, train the VAE every 5k agent steps. Afterward, train
 906 it every 10k steps.
- 907 3. During the first 100k steps of the agent, train the VAE every 2k agent steps. Afterward, train
 908 it every 5k steps.

909 A.2.2 Encoder smooth update

To enhance the stability of the agent’s input representations, actions are selected based on a smoothly
 updated version of the VAE encoder, denoted by parameters $\hat{\phi}$. This encoder is refreshed after each
 VAE training phase and used to produce latent states:

$$a_t = \pi_{\theta}(\cdot | z_t, z_g), \quad z_t = q_{\hat{\phi}}(x_t)$$



Figure 6: Example of decoded pixel goals in fetch environment: we sample latent goals from the latent prior $z \sim p(z) = \mathcal{N}(0, I)$ and plot the corresponding decoded pixel goals $p_\theta(x|z)$. Images were obtained using the decoder trained with DRAG.

910 Analogous to the use of a target network for Q -function updates in RL, the delayed encoder $q_{\hat{\phi}}$ is
 911 updated using an exponential moving average (EMA) of the primary encoder’s weights q_ϕ :

$$\hat{\phi} \leftarrow \tau \hat{\phi} + (1 - \tau) \phi \quad (6)$$

912 Figure 7 illustrates how the smoothing coefficient τ influences success coverage with DRAG. We
 913 observe that $\tau = 0.05$ provides a good balance, yielding stable performance. In contrast, setting
 914 $\tau = 1$ (no smoothing) leads to less stability, while $\tau = 0.01$ results in updates that are too slow. This
 915 value was observed to provide the best average results for other approaches (i.e., RIG and SKEW-FIT).
 916 We use it in any experiment reported in other sections of this paper.

917 A.3 Methods Hyper-parameters

918 The hyper-parameters of our DRAG algorithm are given in Table 2. Notations refer to those used
 919 in the main paper or the pseudo-code given in Algorithm 1. Hyper-parameters that are common to
 920 any approach were set to provide best average results for RIG. RIG, SKEW-FIT and DRAG share
 921 the same values for these hyper-parameters. The skewing temperature for SKEW-FIT, which is not
 922 reported in the tables below, is set to $\alpha = -1$. This value was tuned following a grid search for
 923 $\alpha \in [-100, -50, -10, -5, -1, -0.5, -0.1]$.

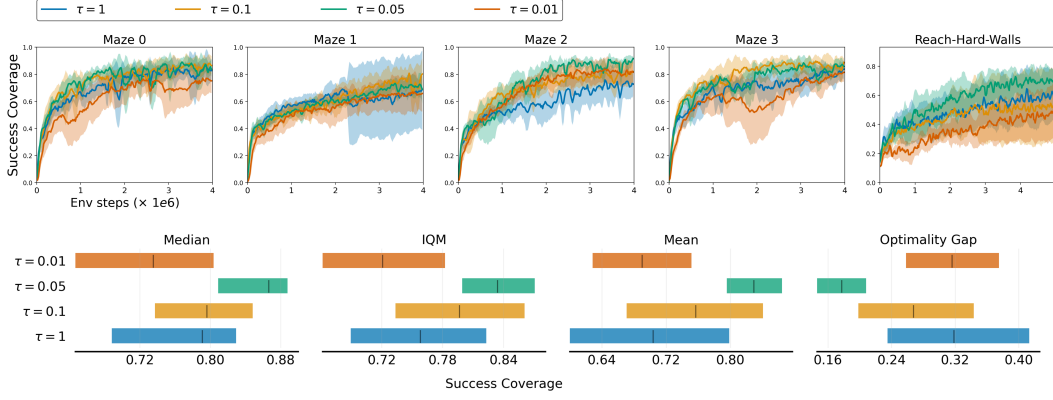


Figure 7: Impact of the exponential moving average smoothing coefficient (τ in Equation (6)) experiment on success coverage (6 seeds per run).

Table 1: Hyper-parameters used for the VAE used in the experiments (same for every approach, top) and values used specifically in DRAG, for the specification of our DRO weighter (bottom).

VAE Hyper-Parameters	Symbol	Value
β-VAE		
Latent dim	d	[Maze env :2, Fetch: 3]
Prior distribution	$p(z)$	$\mathcal{N}(0, I_d)$
Regularization factor	β	2
Learning rate	ϵ	1e-3
CNN channels		(3, 32, 64, 128, 256)
Dense layers		(512,128)
Activation Function		ReLu
CNN Kernel size		4
Training batch size	n	100
Optimization interval (in agent steps)	$freqOpt$	10e3
Nb of training steps	$nbEpochs$	50
Size of training buffer	$ \mathcal{R} $	1e6
Number of samples per epoch	N	1e3
DRO Weighter r_ψ		
Learning rate	ϵ	2e-6
Convolutional layers channels		(3, 32, 64, 128, 256)
Dense layers		(512,128)
Activation Function		ReLu
Temperature	λ	0.01

Table 2: Off-policy RL algorithm TQC parameters

TQC Hyper-Parameters	Value
Batch size for replay buffer	2000
Discount factor γ	0.98
Action L2 regularization	0.1
(Gaussian) Action noise max std	0.1
Warm up steps before training	2500
Actor learning rate	1e-3
Critic learning rate	1e-3
Target network soft update rate	0.05
Actor & critic networks activation	ReLU
Actor & critic hidden layers sizes	512 ³
Replay buffer size ($ \mathcal{B} $)	1e6

Table 3: goal criterion hyper-parameters

Goal criterion Hyper-parameters	Symbol	Value
Kernel density Estimation for MEGA and LGE		
RBF kernel bandwidth	σ	0.1
KDE optimization interval (in agent steps)		1
Nb of state samples for KDE optim.		10,000
Nb of sampled candidate goals from $p(z)$		100
Agent’s skill model D_ϕ for SVGG and GOALGAN		
Hidden layers sizes		(64, 64)
Gradient steps per optimization		100
Learning rate		1e-3
Training batch size		100
Training history length (episodes)		500
Optimization interval (in agent steps)		5000
Nb of training steps		100
Activations		Relu

924 A.4 Compute ressources & code assets

925 This work was performed with 35,000 GPU hours on NVIDIA V100 GPUs (including main experi-
 926 ments and ablations).

927 We use the GCRL library XPAG [Perrin-Gilbert, 2022], designed for intrinsically motivated RL
 928 agents.

929 B DRAG algorithm

930 Algorithm 1 reports the full pseudo-code of our DRAG approach. RIG and SKEW-FIT follow the same
 931 procedure, without the DRO weighter update loop (line 9 to 12), and replacing $\mathcal{L}_{\theta, \phi, r_\psi}^{\text{VAE-DRO}}(\{x_i\}_{i=1}^n)$ in
 932 line 15 by:

- RIG (classical ELBO):

$$\mathcal{L}_{\theta, \phi}^{\text{VAE}}(\{x_i\}_{i=1}^n) = \frac{n}{N} \sum_{i=1}^n \left(\frac{1}{m} \sum_{j=1}^m \log p_\theta(x_i | z_i^j) - KL(q_\phi(z | x_i) || p(z)) \right)$$

- SKEW-FIT:

$$\mathcal{L}_{\theta, \phi}^{\text{VAE-SkewFit}}(\{x_i\}_{i=1}^n) = \frac{n}{N} \sum_{i=1}^n p_{\text{skewed}}(x_i) \left(\frac{1}{m} \sum_{j=1}^m \log p_{\theta}(x_i|z_i^j) - KL(q_{\phi}(z|x_i)||p(z)) \right),$$

933 where $p_{\text{skewed}}(x_i)$ is the skewed distribution of SKEW-FIT, that uses an estimate $\tilde{L}_{\theta, \phi}(x_i)$ of the
 934 generative posterior of x_i from the current VAE, obtained from M codes sampled from $q_{\phi}(z|x_i)$.
 935 More details about p_{skewed} are given in section C.2. For comparison, as a recall, for DRAG we take:

- DRAG:

$$\mathcal{L}_{\theta, \phi, \psi}^{\text{VAE-DRO}}(\{x_i\}_{i=1}^n) = \frac{1}{N} \sum_{i=1}^n r_{\psi}(x_i) \left(\frac{1}{m} \sum_{j=1}^m \log p_{\theta}(x_i|z_i^j) - KL(q_{\phi}(z|x_i)||p(z)) \right),$$

937 where the weighting function r_{ψ} is defined, following equation 2, as: $r_{\psi}(x_i) = n \frac{\exp^{f_{\psi}(x_i)}}{\sum_{j=1}^n \exp^{f_{\psi}(x_j)}}$, for
 938 any minibatch $\{x_i\}_{i=1}^n$.

Algorithm 1 Distributionally Robust Exploration

```

1: Input: a GCP  $\pi_{\theta}$ , a VAE: encoder  $q_{\phi}(z|x)$  and smoothly updated version  $q_{\hat{\phi}}(z|x)$ , decoder
    $p_{\phi}(x|z)$ , latent prior  $p(z)$ , DRO Neural Weighter  $r_{\psi}$ , buffers of transitions  $\mathcal{B}$ , reached states  $\mathcal{R}$ ,
   train size  $N$ , batch-size  $n$ , number  $m$  of sampled noises for each VAE training input, number  $M$ 
   of Monte Carlo samples used to estimate  $\tilde{L}_{\theta, \phi}$  for each input image, temperature  $\lambda$ , number of
   optimization epochs  $nEpochs$ , frequency of VAE and policy optimization  $freqOpt$ .
2: while not stop do
3:    $\triangleright$  Data Collection (during  $freqOpt$  steps): Perform rollouts of  $\pi_{\theta}(\cdot|z_t, z_g)$  in the latent
      space, conditioned on goals sampled from prior  $z_g \sim p(z)$  or the buffer (with possible
      resampling depending on the goal selection strategy), and latent state  $z_t = q_{\hat{\phi}}(x_t)$ , with  $x_t$  a
      pixel observation;
4:   Store transitions in  $\mathcal{B}$ , visited states in  $\mathcal{R}$ ;
5:
6:    $\triangleright$  Learning Representations with VAE
7:   for  $nEpochs$  epochs do
8:     Sample a train set of  $N$  states  $\Gamma$  from  $\mathcal{R}$ 
9:     for every mini-batch  $\{x_i\}_{i=1}^n$  from  $\Gamma$  do
10:       $\triangleright$  DRO Weighter Update
11:      Update weighter by one step of Adam optimizer, for the maximization problem
         from (4) with temperature  $\lambda$ , using  $\tilde{L}_{\theta, \phi}$  estimated from  $M$  samples from  $q_{\phi}(z|x_i)$  for
         each  $x_i$ .
12:     end for
13:     for every mini-batch  $\{x_i\}_{i=1}^n$  from  $\Gamma$  do
14:       $\triangleright$  Weighted VAE Update
15:      Update encoder  $q_{\phi}$  and decoder  $p_{\phi}$  by one step of Adam optimizer on
          $-\mathcal{L}_{\theta, \phi, r_{\psi}}^{\text{VAE-DRO}}(\{x_i\}_{i=1}^n)$ , as defined in (5), with  $m$  sampling noises  $(\epsilon_i^j)_{j=1}^m$  for each  $x_i$ .
16:      Perform smooth update of  $\hat{\phi}$  as a function of  $\phi$  according to equation (6).
17:     end for
18:   end for
19:
20:    $\triangleright$  Agent Improvement
21:   Improve agent with any Off-Policy RL algorithm (e.g., TQC, DDPG, SAC...) using
      transitions from  $\mathcal{B}$ ;
22: end while

```

939 C Skew-Fit is a non-parametric DRO

940 In this section we show that SKEW-FIT is a special case of the non-parametric version of DRO.

941 C.1 Non-parametric solution of DRO

942 We start from the inner maximization problem stated in (1), for a given fixed θ :

$$\begin{aligned} \max_r \quad & \frac{1}{N} \sum_{i=1}^N r(x_i, y_i) l(f_\theta(x_i), y_i) - \lambda \frac{1}{N} \sum_{i=1}^N r(x_i, y_i) \log r(x_i, y_i) \\ \text{st} \quad & \frac{1}{N} \sum_{i=1}^N r(x_i, y_i) = 1. \end{aligned} \quad (7)$$

943 From this formulation, we can see that the risk associated to a shift of test distribution can be mitigated
944 by simply associating adversarial weights $r_i := r(x_i, y_i)$ to every sample (x_i, y_i) from the training
945 dataset, respecting $\bar{r} := \frac{1}{N} \sum_{i=1}^N r_i = 1$. This can be viewed as an infinite capacity function r , able
946 to over-specify on every training data point. Equivalently to (7), we thus have:

$$\begin{aligned} \max_{(r_i)_{i=1}^N} \quad & \frac{1}{N} \sum_{i=1}^N r_i l_i - \lambda \frac{1}{N} \sum_{i=1}^N r_i \log r_i \\ \text{st} \quad & \frac{1}{N} \sum_{i=1}^N r_i = 1, \end{aligned} \quad (8)$$

947 where $l_i := l(f_\theta(x_i), y_i)$. The Lagrangian corresponding to this constrained maximization is given
948 by:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N r_i l_i - \lambda \frac{1}{N} \sum_{i=1}^N r_i \log r_i - \gamma \left(\frac{1}{N} \sum_{i=1}^N r_i - 1 \right) \quad (9)$$

949 where γ is an unconstrained Lagrangian coefficient.

950 Following the Karush-Kuhn-Tucker conditions applied to the derivative of the Lagrangian function \mathcal{L}
951 of this problem in r_i for any given $i \in [[1, N]]$, we obtain:

$$\frac{\partial \mathcal{L}}{\partial r_i} = 0 \Leftrightarrow l_i - \lambda(\log r_i - 1) - \gamma = 0 \Leftrightarrow r_i = e^{\frac{l_i - \gamma}{\lambda} - 1} = z e^{\frac{l_i}{\lambda}} \quad (10)$$

952 with $z := e^{\frac{-\gamma}{\lambda} - 1}$.

The KKT condition on the derivative in γ gives: $\frac{\partial \mathcal{L}}{\partial \gamma} = 0 \Leftrightarrow \frac{1}{N} \sum_{i=1}^N r_i = 1$. Combining these two
results, we thus obtain:

$$\frac{1}{N} \sum_{i=1}^N r_i = \frac{1}{N} \sum_{i=1}^N z e^{\frac{l_i}{\lambda}} = 1 \Leftrightarrow z = \frac{N}{\sum_{i=1}^N e^{\frac{l_i}{\lambda}}}$$

Which again gives, reinjecting this result in Equation (10):

$$r_i = N \frac{e^{\frac{l_i}{\lambda}}}{\sum_{j=1}^N e^{\frac{l_j}{\lambda}}}$$

953 This leads to the form of a Boltzmann distribution, which proves the result.

954 C.2 Application to GCRL with VAE and Relation to Skew-Fit

955 SKEW-FIT resamples training data points from a batch $\{x_i\}_{i=1}^n$ using a skewed distribution defined,
956 for any sample x in that batch, as:

$$\begin{aligned} p_{\text{skewed}}(\mathbf{x}) &\triangleq \frac{1}{Z_\alpha} w_{t,\alpha}(\mathbf{x}), \\ Z_\alpha &= \sum_{i=1}^n w_{t,\alpha}(\mathbf{x}_i), \end{aligned} \quad (11)$$

957 where $w_{t,\alpha}$ is an importance sampling coefficient given as:

$$w_{t,\alpha}(x) := p_{\theta,\phi}(x)^\alpha, \quad \alpha < 0, \quad (12)$$

958 with $p_{\theta,\phi}(x)$ the generative distribution of samples x given current parameters (θ, ϕ) .

Applied to a generative model defined as a VAE, we have:

$$p_{\theta,\phi}(x) = \mathbb{E}_{z \sim q_\phi(z|x)} \frac{p(z)p_\theta(x|z)}{q_\phi(z|x)} dz,$$

959 where $p(z)$ is the prior over latent encodings of the data x , $p_\theta(x|z)$ is the likelihood of x know-
 960 ing z and $q_\phi(z|x)$ the encoding distribution of data points. As stated in Section 3.1, this can
 961 be estimated on a set of m samples for each data point using the log-approximator: $\tilde{\mathcal{L}}_{\theta,\phi}(x) =$
 962 $\log \sum_{j=1}^M \exp(\log p_\theta(x|z^j) + \log p_\theta(z^j) - \log q_\phi(z^j|x)) - \log(M)$.

963 Thus, this is equivalent as associating any i from the data batch with a weight r_i defined as:

$$\begin{aligned} r_i := p_{\text{skewed}}(\mathbf{x}_i) &= \frac{1}{Z_\alpha} e^{\alpha \tilde{\mathcal{L}}_{\theta,\phi}(x_i)}, \\ Z_\alpha &= \sum_{j=1}^n e^{\alpha \tilde{\mathcal{L}}_{\theta,\phi}(x_j)}, \end{aligned} \quad (13)$$

964 for any $\alpha < 0$. Setting $\alpha = -\frac{1}{\lambda}$, we get $p_{\text{skewed}}(\mathbf{x}_i) \propto e^{-\tilde{\mathcal{L}}_{\theta,\phi}(x_i)/\lambda}$, for any temperature $\lambda > 0$.
 965 Reusing the result from Section C.1, this is fully equivalent to the analytical closed-form solution of
 966 DRO when applied to $-\tilde{\mathcal{L}}_{\theta,\phi}(x_i)$ as we use in our DRO-VAE approach. Using p_{skewed} with $\alpha = -\frac{1}{\lambda}$
 967 for weighting training points of a VAE thus exactly corresponds to the non-parametric version our
 968 DRAG algorithm.

969 D Additional results

970 D.1 Visualization of Learned Latent Representations

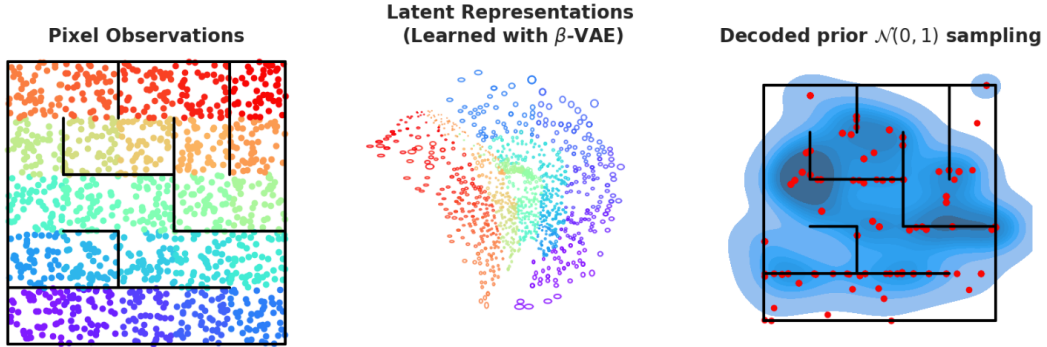


Figure 8: Learned Representation of DRAG after 1 million training steps in Maze 0. **Left:** every colored dot corresponds to the pixel observation x of its specific xy coordinates. **Middle:** Every pixel observation x on the left is processed by the VAE encoder to get the learned latent posterior distribution $q_\phi(z|x) = \mathcal{N}(z|\mu_\phi(x), \sigma_\phi(x))$. Colored ellipsoids correspond to these 2-dimensional Gaussian distributions. **Right:** we sample latent goals from the latent prior $z \sim p(z) = \mathcal{N}(0, I)$ and we decode the corresponding pixel observations $p_\theta(x|z)$ (red dots correspond to the xy coordinates of the pixel observations).

971 Figure 8 presents our methodology to study latent representation learning. We uniformly sample data
 972 points in the maze and process them iteratively from 2D points to pixels, then from pixels to the latent
 973 code of the VAE. Using the same color for the source data points and the latent code, this process
 974 allows us to visualize the 2D latent representation of the VAE in the environment (Figure 8 left and

975 middle). In addition, to get a sense of what part of the environment is encoded in the latent prior,
 976 we sample latent codes from $p(z)$ and plot the 2D coordinates of the decoded observations using
 977 $p_\theta(x|z)$, which corresponds to the red dots. The blue distribution corresponds to a KDE estimate
 978 fitted to the red dots.

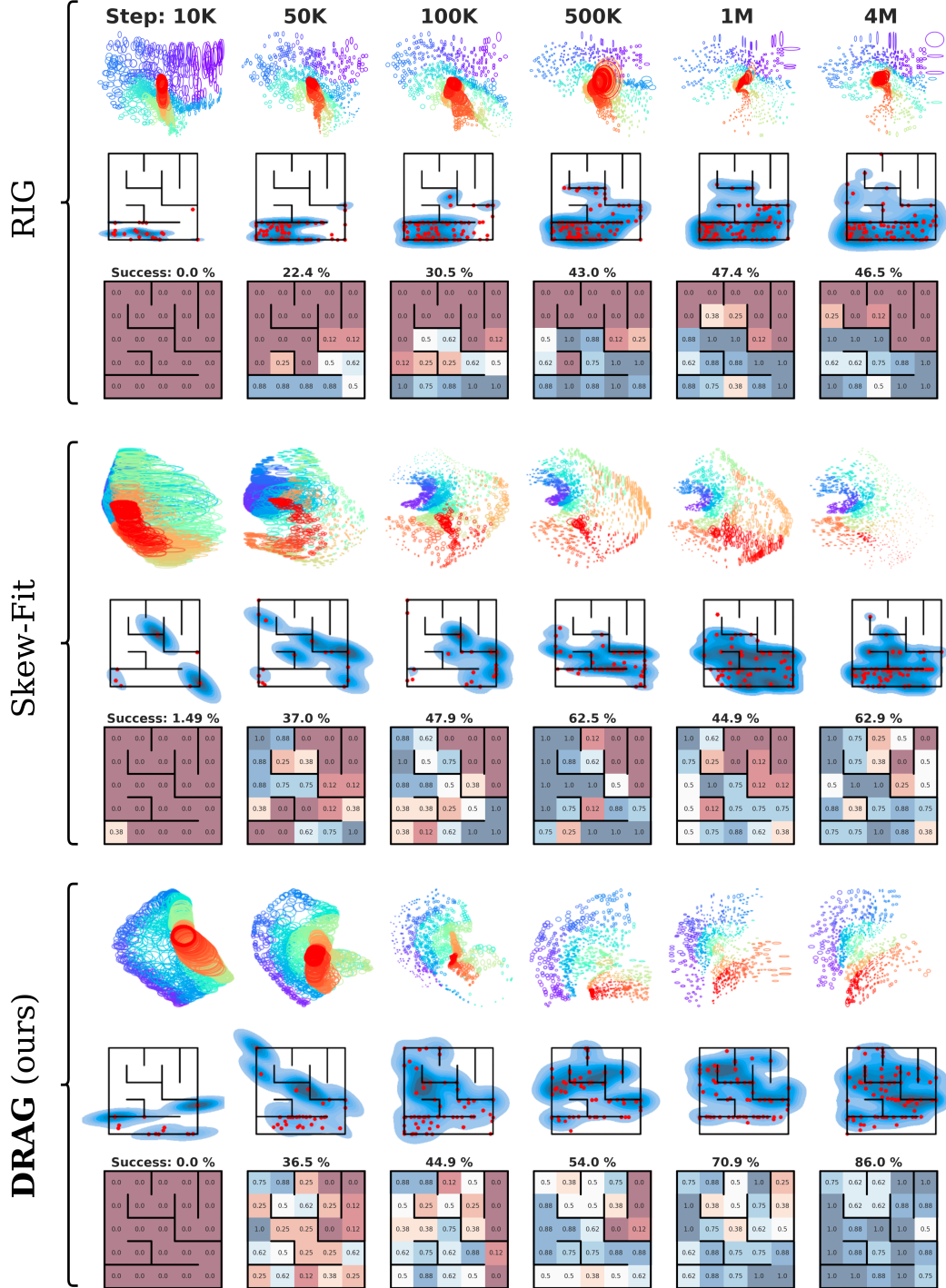


Figure 9: **First row** of each method: evolution of learned representations. **Second row** of each method: evolution of the intrinsic goal distribution when sampling from the latent prior $p(z) = N(0, 1)$. **Third row** of each method: evolution of the success coverage. (See Figure 8 for details on how we obtain these plots).

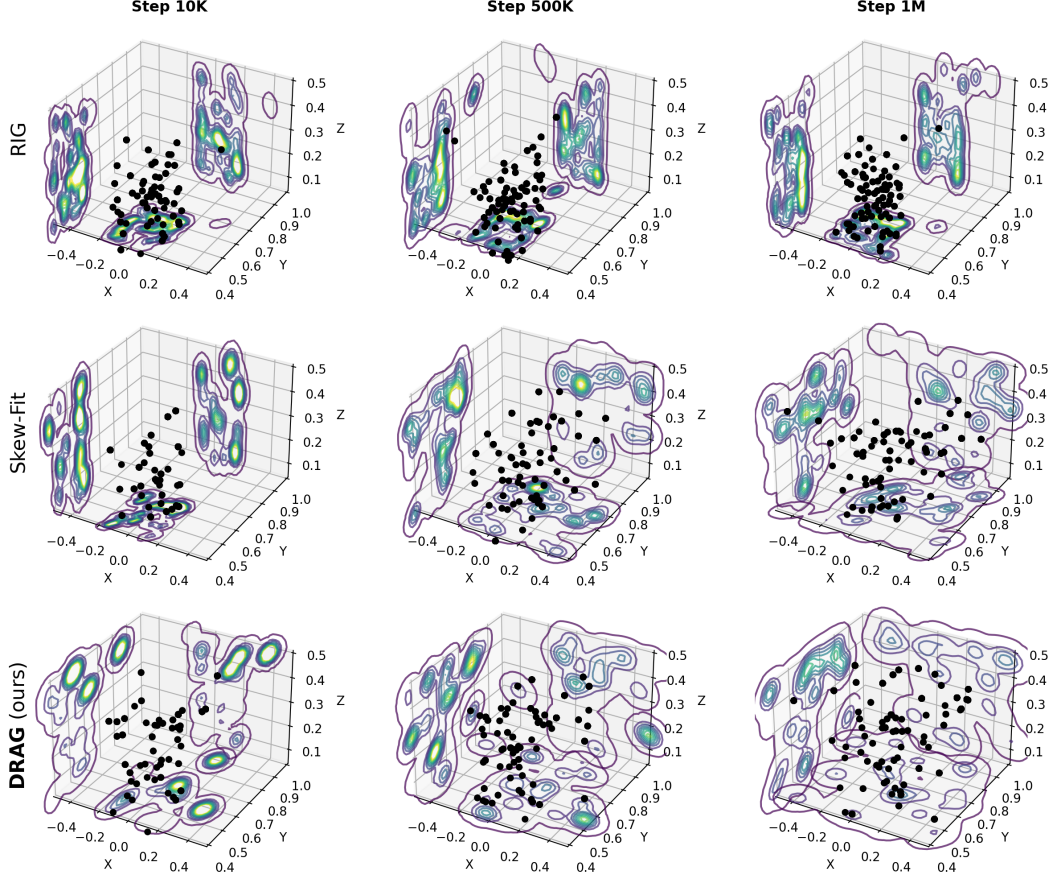


Figure 10: Evolution of the prior distribution in the Fetch environment for the DRAG, SKEW-FIT, and RIG methods: We sample latent goals from the latent prior $z \sim p(z) = \mathcal{N}(0, I)$ and we decode the corresponding pixel observations $p_\theta(x|z)$ (black dots correspond to the 3D xyz coordinates of the pixel observations).

979 In order to gain a deeper insight into the performance of RIG, SKEW-FIT, and DRAG, we show in
 980 Figure 9 and Figure 9 the parallel evolution of the prior sampling $z \sim \mathcal{N}(0, I)$, and the corresponding
 981 learned 2D representations for the maze environment in Figure 9.

982 One can clearly see that RIG is stuck in an exploration bottleneck (which in this case corresponds to
 983 the first U-turn of the maze): the VAE cannot learn meaningful representations of poorly explored
 984 areas (red part of the maze in Figure 9). As a consequence, the prior distribution $p(z)$ only encodes a
 985 small subspace of the environment. On the other hand, SKEW-FIT and DRAG manage to escape these
 986 bottlenecks and incorporate an organized representation of nearly every area of the environment, with
 987 the difference that DRAG is more stable and therefore reliably learns well organized representations.

988 In order to quantify the evolution of latent representations to highlight the differences in terms
 989 of latent distribution dynamics between RIG, SKEW-FIT, and DRAG, we introduce the following
 990 measurement:

$$\forall t = 1 \dots T, \quad d_t(\mathbf{x}) \triangleq \frac{1}{n} \sum_{i=1}^n \|\mu_{\phi^t}(x_i) - \mu_{\phi^{t-1}}(x_i)\|, \quad (14)$$

991 where $\mathbf{x} = \{x_i\}_{i=1}^n$ is a batch of pixel observations uniformly sampled from the environment state
 992 space using prior knowledge (only for evaluation purposes). With this metric, we measure the
 993 evolution of the embedding of every point x_i , using the movement of the expectation $\mu_\phi(x_i)$ from the

latent posterior distribution $q_\phi(z|x_i) = \mathcal{N}(z|\mu_\phi(x_i), \sigma_\phi(x_i))$, throughout updates of VAE parameters ϕ .

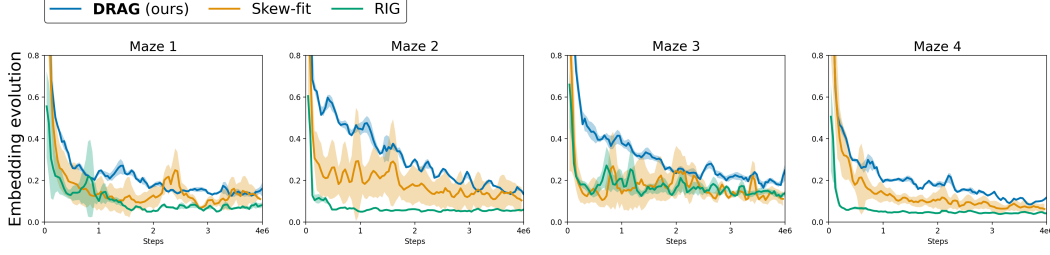


Figure 11: Evolution of the embedding over 4 different PointMazes (6 seeds each) for 4M steps (shaded areas correspond to standard deviation). Every point corresponds to the shift of representation between step t and step $t + 1$ of VAE training: $\frac{1}{n} \sum_{i=1}^n \|\mu_{\phi^{t+1}}(x_i) - \mu_{\phi^t}(x_i)\|$. For every pixel observation x_i and timestep t , we have $q_{\phi^t}(z|x_i) = \mathcal{N}(z|\mu_{\phi^t}(x_i), \sigma_{\phi^t}(x_i))$. We compute representation shifts between t and $t + 1$ every 40,000 training steps.

Figure 11 shows that the embedding movement d of DRAG is higher and less variable across seeds, indicating that the learned representations evolve more consistently. Meanwhile, the VAE training process of SKEW-FIT is prone to variability, and the evolution of the embedding in RIG is close to null after a certain number of training steps.

D.2 Ablations

D.2.1 Impact of λ

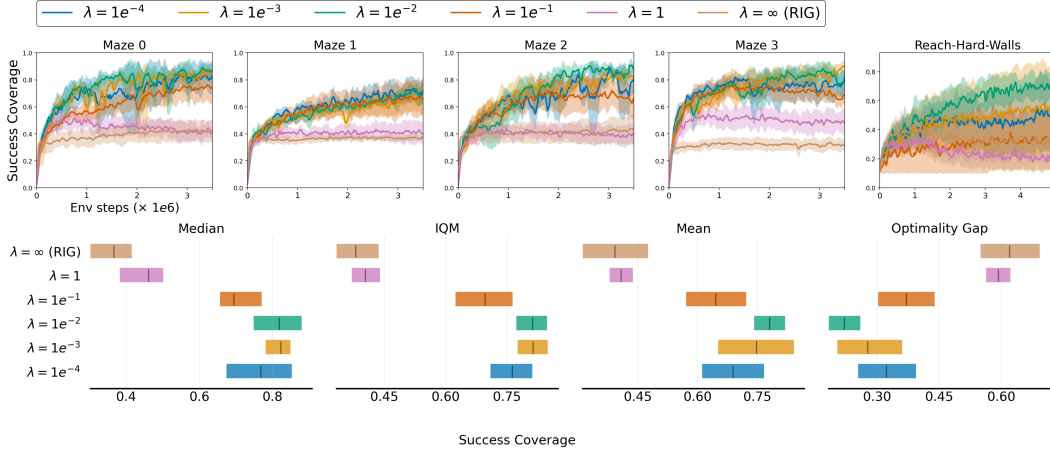


Figure 12: Impact of the regularization parameter λ on the performances of DRAG (6 seeds per run). Results obtained with goals directly selected from the prior (as in Section 4.1). Note that $\lambda = \infty$ comes down to the RIG approach, as weights converge to a constant (over-regularization).

Figure 12 illustrates the effect of varying the regularization parameter λ on the performance of DRAG. As discussed in the main paper, lower values of λ bias the training distribution to emphasize samples from less covered regions of the state space. Conversely, higher values of λ lead to flatter weighting distributions across batches, eventually resembling the behavior of a standard VAE (as used in RIG) when λ becomes very large. In fact, setting $\lambda = \infty$ makes DRAG behave identically to RIG.

The reported results show that DRAG achieves the highest success coverage for λ values between 10 and 100, with a slight edge at $\lambda = 100$. This range represents a good trade-off: too small a λ can lead to unstable training, where the model places excessive weight on underrepresented samples; too large a λ leads to overly strong regularization toward the marginal distribution $p(x)$, limiting generalization, and hence exploration.

For comparison, SKEW-FIT performs best at $\alpha = -1$, which corresponds to $\lambda = 1$ in the DRO (non-parametric) formulation (see Section C for theoretical equivalences). This much lower value reflects a key difference: SKEW-FIT relies on pointwise estimations of the generative posterior, while DRAG uses smoothed estimates provided by a neural weighting function. As a result, SKEW-FIT requires less aggressive skewing to avoid instability.

D.2.2 Impact of M (number of samples for the estimation of $\tilde{L}_{\theta,\phi}$)

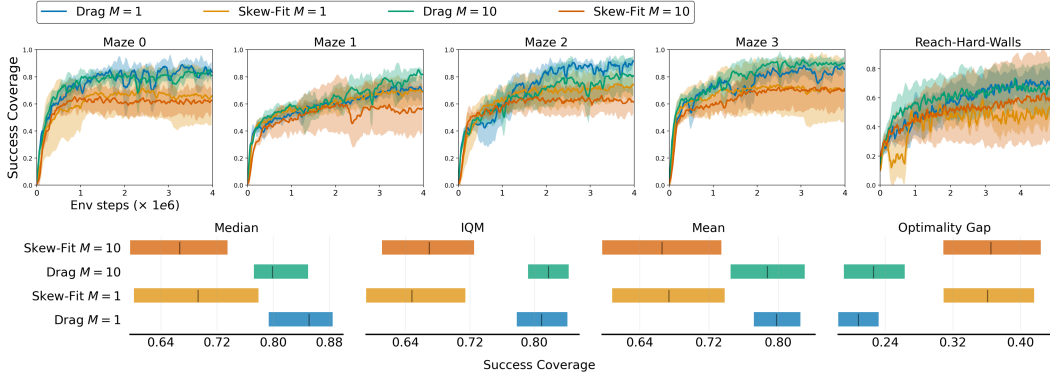


Figure 13: Impact of the number of samples M , used for the estimation of $\tilde{L}_{\theta,\phi}$ in DRAG and SKEW-FIT (6 seeds per run). Results obtained with goals directly selected from the prior (as in section 4.1).

Both SKEW-FIT and DRAG need to estimate the generative posterior of inputs in order to build their VAE weighting schemes. This estimator, denoted in the paper as $\tilde{L}_{\theta,\phi}(x)$ for any input x , is obtained via Monte Carlo samples of codes from $q_\phi(z|x)$. The number M of samples used impacts the variance of this estimator. The higher M is, the more accurate the estimator is, at the cost of an increase of computational resources (M samples means M likelihood computations through the decoder). This section inspects the impact of M on the overall performance.

Figure 13 presents the results for SKEW-FIT and DRAG using $M = 1$ (as in the rest of the paper) and $M = 10$. While one might expect more accurate estimates of $\tilde{L}_{\theta,\phi}(x)$ with $M = 10$, this improvement does not translate into better success coverage for the agent. According to the reported results, the value of M does not appear to significantly impact the agent’s performance for either algorithm. In fact, on average, increasing M even slightly decreases success coverage.

This is a noteworthy finding, as it suggests that the improved stability of DRAG compared to SKEW-FIT is not due to more accurate pointwise likelihood estimation (which could benefit DRAG through the inertia introduced by using a parametric predictor), but rather due to greater spatial smoothness. This smoothness arises from the L -Lipschitz continuity of the neural network: inputs located in the same region of the visual space are assigned similar weights by DRAG’s neural weighting function. In contrast, SKEW-FIT may overemphasize specific inputs, with abrupt weighting shifts, particularly when those inputs are poorly represented in the latent space, despite being located in familiar visual regions.

D.3 RIG+Goal selection criterion

Figure 14 presents the results of combining the RIG representation learning strategy (i.e., without biasing VAE training) with a goal selection criterion, following the same experimental setup as in Figure 2b. These results highlight two key insights.

First, the results show that the exploration limitations inherent to the RIG strategy cannot be effectively addressed by improved goal selection alone. Even the best-performing combination (RIG + LGE) achieves less than 60% success coverage on average across environments, while DRAG alone reaches 80%. This highlights the critical role of DRAG’s representation learning in overcoming exploration bottlenecks in complex environments. When relying solely on the latent space of a standard VAE, sampling—even when guided by intrinsic motivation—remains limited to regions already well-

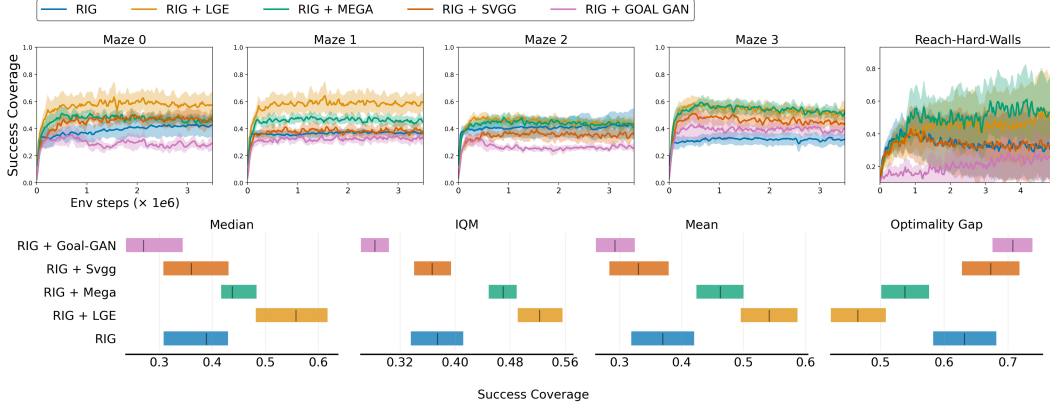


Figure 14: Impact of goal resampling with classical (unbiased) VAE training, as in RIG. Evolution of the success coverage for different goal sampling methods (6 seeds per run). RIG directly uses goals sampled from the prior (i.e., same results as RIG in figure 3), RIG + X includes an additional goal resampling method X, taken among the four strategies: LGE, MEGA, GOALGAN or SVGG. This figure presents the same experiment as in Figure 4, but using a standard VAE instead of our proposed DRO-VAE.

1047 represented in the training data. The model cannot generate goals in poorly explored areas, as no
 1048 latent codes exist that decode to such states.

1049 Second, we observe a reversal in the relative effectiveness of goal selection criteria compared to
 1050 the DRAG experiments in Figure 2b. In the case of RIG, both LGE and MEGA outperform SVGG,
 1051 which contrasts with the pattern observed with DRAG. This can be explained by the fact that RIG
 1052 is inherently limited in its ability to explore, and thus benefits more from goal selection strategies
 1053 that explicitly promote exploration. In contrast, strategies based on intermediate difficulty, such as
 1054 SVGG, are less effective when the agent is confined to a limited region of the environment. Latent
 1055 codes associated with intermediate difficulty typically decode to well-known states, while those
 1056 corresponding to poorly explored areas often lead to posterior distributions with higher variance. As
 1057 a result, the latter are more likely to be classified as too difficult and filtered out. These strategies
 1058 therefore tend to reinforce learning around familiar areas, without actively pushing the agent toward
 1059 under-explored or novel regions that are critical for improving coverage. This type of goal selection
 1060 can therefore only be effective when built on top of representations—such as the one learned by
 1061 DRAG—that are explicitly encouraged to include marginal or rarely visited states.

1062 E Limitations

1063 **Latent space reward definition** While our study makes progress on learning representations online
 1064 and generating intrinsic goals from high-dimensional observations, it does not address how to measure
 1065 when a goal has truly been achieved. Throughout our experiments, we relied on a simple sparse
 1066 reward $r_t = \mathbb{1}[||z_{x_t} - z_g||_2 < \delta]$ which, although common in goal-conditioned RL, sidesteps the
 1067 challenges of defining a dense feedback signal. In particular, the Euclidean metric used in dense
 1068 rewards often fails to reflect the true topology of the environment and can mislead the agent.

1069 **Representation Learning algorithm** Our DRO-based approach is agnostic to the choice of repre-
 1070 sentation learning algorithm, suggesting future work should benchmark alternatives such as other
 1071 reconstruction-based techniques [Van Den Oord et al., 2017, Razavi et al., 2019, Gregor et al., 2019],
 1072 or contrastive learning objectives [Oord et al., 2018, Henaff, 2020, He et al., 2020, Zbontar et al.,
 1073 2021].

1074 **Leveraging Pre-trained Representations** Our study did not leverage pre-trained visual representa-
 1075 tions, that could greatly improve performance on complex visual observations as demonstrated in
 1076 [Zhou et al., 2025]. In particular, future work should explore incorporating into our setting pre-trained
 1077 representations from models specific to RL tasks as VIP [Ma et al., 2022] and R3M [Nair et al., 2022]

1078 as well as general-purpose visual encoders such as CLIP [Radford et al., 2021] or DINO models
1079 [Caron et al., 2021, Oquab et al., 2024].