A. Proofs

The definitions of continuity and convergence for pseudo-metrics are similar to those for metrics, and we state them below.

A function $f : Q \rightarrow Q$ is continuous with respect to a pseudo-metric $d$ if for any $p \in Q$ and any scalar $\epsilon > 0$, there exists a $\delta$ such that for all $q \in Q$,

$$d(p, q) < \delta \implies d(f(p), f(q)) < \epsilon.$$ 

An infinite sequence $p_1, p_2 \ldots$ converges to a value $p$ with respect to a pseudo-metric $d$, which we write as

$$\lim_{t \to \infty} p_t \to p,$$

if

$$\lim_{t \to \infty} d(p_t, p) \to 0.$$ 

Note that if $f$ is continuous, then

$$\lim_{t \to \infty} d_H(p_t, q) \to 0 \implies \lim_{t \to \infty} d_H(f(p_t), f(q)) \to 0.$$ 

A.1. Proof of Lemma 3.1

**Lemma A.1.** Let $S$ be a compact set. Define the set of distributions $Q = \{p : \text{support}(p) \subseteq S\}$. Let $F : Q \rightarrow Q$ be continuous with respect to the pseudometric $d_H(p, q) \equiv |H(p) - H(q)|$ and $H(F(p)) \geq H(p)$ with equality if and only if $p$ is the uniform probability distribution on $S$, denoted as $U_S$. Define the sequence of distributions $P = (p_1, p_2, \ldots)$ by starting with any $p_1 \in Q$ and recursively defining $p_{t+1} = F(p_t)$. The sequence $P$ converges to $U_S$ with respect to $d_H$. In other words, $\lim_{t\to0} |H(p_t) - H(U_S)| \to 0$.

**Proof.** The idea of the proof is to show that the distance (with respect to $d_H$) between $p_t$ and $U_S$ converges to a value. If this value is 0, then the proof is complete since $U_S$ uniquely has zero distance to itself. Otherwise, we will show that this implies that $F$ is not continuous, which is a contradiction.

For shorthand, define $d_t$ to be the $d_H$-distance to the uniform distribution, as in

$$d_t \equiv d_H(p_t, U_S).$$ 

First we prove that $d_t$ converges. Since the entropies of the sequence $(p_1, \ldots)$ monotonically increase, we have that

$$d_1 \geq d_2 \geq \ldots.$$ 

We also know that $d_t$ is lower bounded by 0, and so by the monotonic convergence theorem, we have that

$$\lim_{t \to \infty} d_t \to d^*,$$

for some value $d^* \geq 0$.

To prove the lemma, we want to show that $d^* = 0$. Suppose, for contradiction, that $d^* \neq 0$. Then consider any distribution, $q^*$, such that $d_H(q^*, U_S) = d^*$. Such a distribution always exists since we can continuously interpolate entropy values between $H(p_1)$ and $H(U_S)$ with a mixture distribution. Note that $q^* \neq U_S$ since $d_H(U_S, U_S) = 0$. Since $\lim_{t \to \infty} d_t \to d^*$, we have that

$$\lim_{t \to \infty} d_H(p_t, q^*) \to 0,$$

and so

$$\lim_{t \to \infty} p_t \to q^*.$$ 

Because the function $F$ is continuous with respect to $d_H$, Equation 5 implies that

$$\lim_{t \to \infty} d_H(F(p_t), F(q^*)) \to 0.$$ 

However, since $F(p_t) = p_{t+1}$ we can equivalently write the above equation as

$$\lim_{t \to \infty} d_H(p_{t+1}, F(q^*)) \to 0.$$ 

which, through a change of index variables, implies that

$$\lim_{t \to \infty} p_t \to F(q^*)$$ 

Since $q^*$ is not the uniform distribution, we have that $H(F(q^*)) > H(q^*)$, which implies that $F(q^*)$ and $q^*$ are unique distributions. So, $p_t$ converges to two distinct values, $q^*$ and $F(q^*)$, which is a contradiction. Thus, it must be the case that $d^* = 0$, completing the proof.

A.2. Proof of Lemma 3.2

**Lemma A.2.** Given two distribution $p(x)$ and $q(x)$ where $p \ll q$ and

$$0 < \text{Cov}_p[\log p(X), \log q(X)]$$

define the distribution $p_\alpha$ as

$$p_\alpha(x) = \frac{1}{Z_\alpha} p(x) q(x)^\alpha$$

where $\alpha \in \mathbb{R}$ and $Z_\alpha$ is the normalizing factor. Let $H_\alpha(\alpha)$ be the entropy of $p_\alpha$. Then there exists a constant $a > 0$ such that for all $\alpha \in [-a, 0)$,

$$H_\alpha(\alpha) > H_\alpha(0) = H(p).$$


The sequence $\{p_\alpha : \alpha \in [-1, 0]\}$ is a one-dimensional exponential family

$$p_\alpha(x) = e^{\alpha T(x) - A(\alpha) + k(x)}$$

with log carrier density $k(x) = \log p(x)$, natural parameter $\alpha$, sufficient statistic $T(x) = \log q(x)$, and log-normalizer $A(\alpha) = \int_x e^{\alpha T(x) + k(x)} dx$. As shown in (Nielsen & Nock, 2010), the entropy of a distribution from a one-dimensional exponential family with parameter $\alpha$ is given by:

$$H_\alpha(x) \triangleq H(p_\alpha) = A(\alpha) - \alpha A'(\alpha) - E_{p_\alpha}[k(X)]$$

The derivative with respect to $\alpha$ is then

$$\frac{d}{d\alpha} H_{\alpha}(\alpha) = -\alpha A''(\alpha) - \frac{d}{d\alpha} E_{p_\alpha}[k(x)]$$

$$= -\alpha A''(\alpha) - E_{p_\alpha}[k(x)](T(x) - A'(\alpha)]$$

$$= -\alpha \text{Var}_{p_\alpha}(T(x)) - \text{Cov}_{p_\alpha}(k(x), T(x))$$

where we use the fact that the $n$th derivative of $A(\alpha)$ give the $n$ central moment, i.e. $A'('(\alpha) = E_{p_\alpha}[T(x)]$ and $A''(\alpha) = \text{Var}_{p_\alpha}[T(x)]$. The derivative of $\alpha = 0$ is

$$\frac{d}{d\alpha} H_{\alpha}(0) = -\text{Cov}_{p}[k(x), T(x)]$$

$$= -\text{Cov}_{p}[\log p(x), \log q(x)]$$

which is negative by assumption. Because the derivative at $\alpha = 0$ is negative, then there exists a constant $\alpha > 0$ such that for all $\alpha \in [-\alpha, 0]$, $H_\alpha(\alpha) > H_\alpha(0) = H(p)$. □

The paper applies to the case where

$$\text{Sensitivity to RL Algorithm}$$

In our experiments, we combined Skew-Fit with soft actor critic (SAC) (Haarnoja et al., 2018). We conduct a set of experiments to test whether Skew-Fit may be used with other RL algorithms for training the goal-conditioned policy. To that end, we replaced
Skew-Fit: State-Covering Self-Supervised Reinforcement Learning

We compare using SAC (Haarnoja et al., 2018) and TD3 (Fujimoto et al., 2018) as the underlying RL algorithm on Visual Door, Visual Pusher and Visual Pickup. We see that Skew-Fit works consistently well with both SAC and TD3, demonstrating that Skew-Fit may be used with various RL algorithms. For the experiments presented in Section 6, we used SAC.

Sensitivity to $\alpha$ Hyperparameter We study the sensitivity of the $\alpha$ hyperparameter by testing values of $\alpha \in [-1, -0.75, -0.5, -0.25, 0]$ on the Visual Door and Visual Pusher task. The results are included in Figure 10 and shows that our method is robust to different parameters of $\alpha$, particularly for the more challenging Visual Pusher task. Also, the method consistently outperform $\alpha = 0$, which is equivalent to sampling uniformly from the replay buffer.

B.2. Variance Ablation

We measure the gradient variance of training a VAE on an unbalanced Visual Door image dataset with Skew-Fit vs Skew-Fit with importance sampling (IS) vs no Skew-Fit (labeled MLE). We construct the imbalanced dataset by rolling out a random policy in the environment and collecting the visual observations. Most of the images contained the door in a closed position; in a few, the door was opened. In Figure 11, we see that the gradient variance for Skew-Fit with IS is catastrophically large for large values of $\alpha$. In contrast, for Skew-Fit with SIR, which is what we use in practice, the
Table 1. Despite training on an unbalanced Visual Door dataset (see Figure 7 of paper), the negative log-likelihood (NLL) of Skew-Fit evaluated on a uniform dataset matches that of a VAE trained on a uniform dataset.

We estimate the density under the VAE by using a sample-wise approximation to the marginal over $x$ estimated using importance sampling:

$$q^G_{\phi_t}(x) = E_{z \sim q_{th}(z|x)} \left[ \frac{p(z)}{q_{th}(z|x)} p_{\psi_t}(x | z) \right]$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \left[ \frac{p(z)}{q_{th}(z|x)} p_{\psi_t}(x | z) \right].$$

where $q_{th}$ is the encoder, $p_{\psi_t}$ is the decoder, and $p(z)$ is the prior, which in this case is unit Gaussian. We found that sampling $N = 10$ latents for estimating the density worked well in practice.

C.2. Oracle 2D Navigation Experiments

We initialize the VAE to the bottom left corner of the environment for Four Rooms. Both the encoder and decoder have 2 hidden layers with [400, 300] units, ReLU hidden activations, and no output activations. The VAE has a latent dimension of 8 and a Gaussian decoder trained with a fixed variance, batch size of 256, and 1000 batches at each iteration. The VAE is trained on the exploration data buffer every 1000 rollouts.

C.3. Implementation of SAC and Prior Work

For all experiments, we trained the goal-conditioned policy using soft actor critic (SAC) (Haarnoja et al., 2018). To make the method goal-conditioned, we concatenate the target XY-goal to the state vector. During training, we retroactively relabel the goals (Kaelbling, 1993; Andrychowicz et al., 2017) by sampling from the goal distribution with probability 0.5. Note that the original RIG (Nair et al., 2018) paper used TD3 (Fujimoto et al., 2018), which we also replaced with SAC in our implementation of RIG. We found that maximum entropy policies in general improved the performance of RIG, and that we did not need to add noise on top of the stochastic policy’s noise. In the prior RIG method, the VAE was pre-trained on a uniform sampling of images from the state space of each environment. In order to ensure a fair comparison to Skew-Fit, we forego pre-training and instead train the VAE alongside RL, using the variant described in the RIG paper. For our RL network architectures and training scheme, we use fully connected networks for the policy, Q-function and value networks with two hidden layers of size 400 and 300 each. We also delay training any of these networks for 10000 time steps in order to collect sufficient data for the replay buffer as well as to ensure the latent space of the VAE is relatively stable (since we continuously train the VAE concurrently with RL training). As in RIG, we train a goal-conditioned value functions (Schaul et al., 2015) using hindsight experience replay (Andrychowicz et al., 2017), relabelling 50% of exploration goals as goals sampled from the VAE prior $N(0, 1)$ and 30% from future goals in the trajectory.

For our implementation of (Hazan et al., 2019), we trained the policies with the reward

$$r(s) = r_{\text{Skew-Fit}}(s) + \lambda \cdot r_{\text{Hazan et al.}}(s)$$

For $r_{\text{Hazan et al.}}$, we use the reward described in Section 5.2 of Hazan et al. (2019), which requires an estimated likelihood of the state. To compute these likelihood, we use the same method as in Skew-Fit (see Appendix C.1). With 3 seeds each, we tuned $\lambda$ across values [100, 10, 1, 0.1, 0.01, 0.001] for the door task, but all values performed poorly. For the pushing and picking tasks, we tested values across [1, 0.1, 0.01, 0.001, 0.0001] and found that 0.1 and 0.01 performed best for each task, respectively.

C.4. RIG with Skew-Fit Summary

Algorithm 2 provides detailed pseudo-code for how we combined our method with RIG. Steps that were removed from the base RIG algorithm are highlighted in blue and
Figure 12. Proposed goals from the VAE for RIG and with Skew-Fit on the Visual Pickup, Visual Pusher, and Visual Door environments. Standard RIG produces goals where the door is closed and the object and puck is in the same position, while RIG + Skew-Fit proposes goals with varied puck positions, occasional object goals in the air, and both open and closed door angles.
Figure 13. Proposed goals from the VAE for RIG (left) and with RIG + Skew-Fit (right) on the *Real World Visual Door* environment. Standard RIG produces goals where the door is closed while RIG + Skew-Fit proposes goals with both open and closed door angles.

Figure 14. Example reached goals by Skew-Fit and RIG. The first column of each environment section specifies the target goal while the second and third columns show reached goals by Skew-Fit and RIG. Both methods learn how to reach goals close to the initial position, but only Skew-Fit learns to reach the more difficult goals.
We found that initially training the V AE without Skew-Fit made little sense to use those estimates to prioritize goals early on in training. Instead, we simply train using MLE training for the first 5K timesteps, and after that we perform Skew-Fit according to the VAE schedules above. Table 2 lists the hyper-parameters that were shared across the continuous control experiments. Table 3 lists hyper-parameters specific to each environment. Additionally, Appendix C.4 discusses the combined RIG + Skew-Fit algorithm.

### Algorithm 2 RIG and RIG + Skew-Fit

**Require:** \( \beta \)-VAE mean encoder \( q_{\mu} \), \( \beta \)-VAE decoder \( p_{\phi} \), policy \( \pi_{\theta} \), goal-conditioned value function \( Q_{w} \), skew parameter \( \alpha \), VAE Training Schedule.

1. Collect \( D = \{ s^{(i)} \} \) using random initial policy.
2. Train \( \beta \)-VAE on data uniformly sampled from \( D \).
3. Fit prior \( p(z) \) to latent encodings \( \{ \mu_{\phi}(s^{(i)}) \} \).
4. for \( n = 0, \ldots, N - 1 \) episodes do
5. Sample latent goal from prior \( z_{g} \sim p(z) \).
6. Sample state \( s_{g} \sim p_{skewed}(s_{g}) \) and encode \( z_{g} = q_{\phi}(s_{g}) \) if \( R \) is nonempty. Otherwise sample \( z_{g} \sim p(z) \).
7. Sample initial state \( s_{0} \) from the environment.
8. for \( t = 0, \ldots, H - 1 \) steps do
9. Get action \( a_{t} \sim \pi_{\theta}(q_{\phi}(s_{t}), z_{g}) \).
10. Get next state \( s_{t+1} \sim p(\cdot | s_{t}, a_{t}) \).
11. Store \( (s_{t}, a_{t}, s_{t+1}, z_{g}) \) into replay buffer \( R \).
12. Sample transition \( (s, a, s', z_{g}) \sim R \).
13. Encode \( \hat{z} = q_{\phi}(s), z^{*} = q_{\phi}(s') \).
14. (Probability 0.5) replace \( z_{g} \) with \( z^{*} \sim p(z) \).
15. (Probability 0.5) replace \( z_{g} \) with \( q_{\phi}(z'') \) where \( z'' \sim p_{skewed} \).
16. Compute new reward \( r = ||z' - z_{g}|| \).
17. Minimize Bellman Error using \( (z, a, z', z_{g}, r) \).
18. end for
19. for \( t = 0, \ldots, H - 1 \) steps do
20. for \( i = 0, \ldots, k - 1 \) steps do
21. Sample future state \( s_{h_{i}}, t < h_{i} \leq H - 1 \).
22. Store \( (s_{t}, a_{t}, s_{t+1}, q_{\phi}(s_{h_{i}})) \) into \( R \).
23. end for
24. end for
25. Construct skewed replay buffer distribution \( p_{skewed}^{n+1} \) using data from \( R \) with Equation 3.
26. if total steps \( < 5000 \) then
27. Fine-tune \( \beta \)-VAE on data uniformly sampled from \( R \) according to VAE Training Schedule.
28. else
29. Fine-tune \( \beta \)-VAE on data uniformly sampled from \( R \) according to VAE Training Schedule.
30. Fine-tune \( \beta \)-VAE on data sampled from \( p_{skewed}^{n+1} \) according to VAE Training Schedule.
31. end if
32. end for

### D. Environment Details

**Four Rooms:** A 20 x 20 2D pointmass environment in the shape of four rooms (Sutton et al., 1999). The observation is the 2D position of the agent, and the agent must specify a target 2D position as the action. The dynamics of the environment are the following: first, the agent is teleported to the target position, specified by the action. Then a Gaussian change in position with mean 0 and standard deviation 0.0605 is applied. If the action would result in the agent

---

4In the main paper, we rounded this to 0.06, but this difference does not matter.
Skew-Fit: State-Covering Self-Supervised Reinforcement Learning

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td># training batches per time step</td>
<td>2</td>
<td>Marginal improvements after 2</td>
</tr>
<tr>
<td>Exploration Noise</td>
<td>None</td>
<td>Did not tune</td>
</tr>
<tr>
<td>RL Batch Size</td>
<td>1024</td>
<td>smaller batch sizes work as well</td>
</tr>
<tr>
<td>VAE Batch Size</td>
<td>64</td>
<td>Did not tune</td>
</tr>
<tr>
<td>Discount Factor</td>
<td>0.99</td>
<td>Did not tune</td>
</tr>
<tr>
<td>Reward Scaling</td>
<td>1</td>
<td>Did not tune</td>
</tr>
<tr>
<td>Policy Hidden Sizes</td>
<td>[400, 300]</td>
<td>Did not tune</td>
</tr>
<tr>
<td>Policy Hidden Activation</td>
<td>ReLU</td>
<td>Did not tune</td>
</tr>
<tr>
<td>Q-Function Hidden Sizes</td>
<td>[400, 300]</td>
<td>Did not tune</td>
</tr>
<tr>
<td>Q-Function Hidden Activation</td>
<td>ReLU</td>
<td>Did not tune</td>
</tr>
<tr>
<td>Replay Buffer Size</td>
<td>100000</td>
<td>Did not tune</td>
</tr>
<tr>
<td>Number of Latents for Estimating Density (N)</td>
<td>10</td>
<td>Marginal improvements beyond 10</td>
</tr>
</tbody>
</table>

Table 2. General hyper-parameters used for all visual experiments.

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>Visual Pusher</th>
<th>Visual Door</th>
<th>Visual Pickup</th>
<th>Real World Visual Door</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path Length</td>
<td>50</td>
<td>100</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>$\beta$ for $\beta$-VAE</td>
<td>20</td>
<td>20</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>Latent Dimension Size</td>
<td>4</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>$\alpha$ for Skew-Fit</td>
<td>−1</td>
<td>−1/2</td>
<td>−1</td>
<td>−1/2</td>
</tr>
<tr>
<td>VAE Training Schedule</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Sample Goals From</td>
<td>$q_{G\phi}^{G}$</td>
<td>$p_{skewed}$</td>
<td>$p_{skewed}$</td>
<td>$p_{skewed}$</td>
</tr>
</tbody>
</table>

Table 3. Environment specific hyper-parameters for the visual experiments.

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># training batches per time step</td>
<td>.50625</td>
</tr>
<tr>
<td>Exploration Noise</td>
<td>None (SAC policy is stochastic)</td>
</tr>
<tr>
<td>RL Batch Size</td>
<td>512</td>
</tr>
<tr>
<td>VAE Batch Size</td>
<td>64</td>
</tr>
<tr>
<td>Discount Factor</td>
<td>299</td>
</tr>
<tr>
<td>Reward Scaling</td>
<td>10</td>
</tr>
<tr>
<td>Path length</td>
<td>300</td>
</tr>
<tr>
<td>Policy Hidden Sizes</td>
<td>[400, 300]</td>
</tr>
<tr>
<td>Policy Hidden Activation</td>
<td>ReLU</td>
</tr>
<tr>
<td>Q-Function Hidden Sizes</td>
<td>[400, 300]</td>
</tr>
<tr>
<td>Q-Function Hidden Activation</td>
<td>ReLU</td>
</tr>
<tr>
<td>Replay Buffer Size</td>
<td>1000000</td>
</tr>
<tr>
<td>Number of Latents for Estimating Density (N)</td>
<td>10</td>
</tr>
<tr>
<td>$\beta$ for $\beta$-VAE</td>
<td>10</td>
</tr>
<tr>
<td>Latent Dimension Size</td>
<td>2</td>
</tr>
<tr>
<td>$\alpha$ for Skew-Fit</td>
<td>−2.5</td>
</tr>
<tr>
<td>VAE Training Schedule</td>
<td>3</td>
</tr>
<tr>
<td>Sample Goals From</td>
<td>$p_{skewed}$</td>
</tr>
</tbody>
</table>

Table 4. Hyper-parameters used for the ant experiment.
moving through or into a wall, then the agent will be stopped at the wall instead.

**Ant:** A MuJoCo (Todorov et al., 2012) ant environment. The observation is a 3D position and velocities, orientation, joint angles, and velocity of the joint angles of the ant (8 total). The observation space is 29 dimensions. The goal is a target 2D position, and the reward is the negative Euclidean distance between the achieved 2D position and target 2D position.

**Visual Pusher:** A MuJoCo environment with a 7-DoF Sawyer arm and a small puck on a table that the arm must push to a target position. The agent controls the arm by commanding \( x, y \) position for the end effector (EE). The underlying state is the EE position, \( e \) and puck position \( p \). The evaluation metric is the distance between the goal and final puck positions. The hand goal/state space is a 10x10 cm\(^2\) box and the puck goal/state space is a 30x20 cm\(^2\) box. Both the hand and puck spaces are centered around the origin. The action space ranges in the interval \([-1, 1]\) in the \( x \) and \( y \) dimensions.

**Visual Door:** A MuJoCo environment with a 7-DoF Sawyer arm and a door on a table that the arm must pull open to a target angle. Control is the same as in Visual Pusher. The evaluation metric is the distance between the goal and final door angle, measured in radians. In this environment, we do not reset the position of the hand or door at the end of each trajectory. The state/goal space is a 5x20x15 cm\(^3\) box in the \( x, y, z \) dimensions respectively for the arm and an angle between \([0, 83]\) radians. The action space ranges in the interval \([-1, 1]\) in the \( x, y, z \) dimensions.

**Visual Pickup:** A MuJoCo environment with the same robot as Visual Pusher, but now with a different object. The object is cube-shaped, but a larger intangible sphere is overlaid on top so that it is easier for the agent to see. Moreover, the robot is constrained to move in 2 dimension: it only controls the \( y, z \) arm positions. The \( x \) position of both the arm and the object is fixed. The evaluation metric is the distance between the goal and final object position. For the purpose of evaluation, 75\% of the goals have the object in the air and 25\% have the object on the ground. The state/goal space for both the object and the arm is 10cm in the \( y \) dimension and 13cm in the \( z \) dimension. The action space ranges in the interval \([-1, 1]\) in the \( y \) and \( z \) dimensions.

**Real World Visual Door:** A Rethink Sawyer Robot with a door on a table. The arm must pull the door open to a target angle. The agent controls the arm by commanding the \( x, y, z \) velocity of the EE. Our controller commands actions at a rate of up to 10Hz with the scale of actions ranging up to 1cm in magnitude. The underlying state and goal is the same as in Visual Door. Again we do not reset the position of the hand or door at the end of each trajectory. We obtain images using a Kinect Sensor. The state/goal space for the environment is a 10x10x10 cm\(^3\) box. The action space ranges in the interval \([-1, 1]\) (in cm) in the \( x, y, z \) dimensions. The door angle lies in the range \([0, 45]\) degrees.

### E. Goal-Conditioned Reinforcement Learning Minimizes \( H(G | S) \)

Some goal-conditioned RL methods such as Warde-Farley et al. (2018); Nair et al. (2018) present methods for minimizing a lower bound for \( H(G | S) \), by approximating \( \log p(G | S) \) and using it as the reward. Other goal-conditioned RL methods (Kaelbling, 1993; Lillicrap et al., 2016; Schaul et al., 2015; Andrychowicz et al., 2017; Pong et al., 2018; Florensa et al., 2018a) are not developed with the intention of minimizing the conditional entropy \( H(G | S) \). Nevertheless, one can see that goal-conditioned RL generally minimizes \( H(G | S) \) by noting that the optimal goal-conditioned policy will deterministically reach the goal. The corresponding conditional entropy of the goal given the state, \( H(G | S) \), would be zero, since given the current state, there would be no uncertainty over the goal (the goal must have been the current state since the policy is optimal).

So, the objective of goal-conditioned RL can be interpreted as finding a policy such that \( H(G | S) = 0 \). Since zero is the minimum value of \( H(G | S) \), then goal-conditioned RL can be interpreted as minimizing \( H(G | S) \).