# DisCo RL: Distribution-Conditioned Reinforcement Learning for General-Purpose Policies

**Soroush Nasiriany*, Vitchyr H. Pong*, Ashvin Nair***
**Alexander Khazatsky, Glen Berseth, Sergey Levine**
University of California, Berkeley
{snasiriany,vitchyr,anair17,khazatsky,gberseth,svlevine}@berkeley.edu

## Abstract

Can we use reinforcement learning to learn general-purpose policies that can perform a wide range of different tasks, resulting in flexible and reusable skills? Contextual policies provide this capability in principle, but the representation of the context determines the degree of generalization and expressivity. Categorical contexts preclude generalization to entirely new tasks. Goal-conditioned policies may enable some generalization, but cannot capture all tasks that might be desired. In this paper, we propose goal distributions as a general and broadly applicable task representation suitable for contextual policies. Goal distributions are general in the sense that they can represent any state-based reward function when equipped with an appropriate distribution class, while the particular choice of distribution class allows us to trade off expressivity and learnability. We develop an off-policy algorithm called distribution-conditioned reinforcement learning (DisCo RL) to efficiently learn these policies. We evaluate DisCo RL on a variety of robot manipulation tasks and find that it significantly outperforms prior methods on tasks that require generalization to new goal distributions.

## 1 Introduction

Versatile, general-purpose robotic systems will require not only broad repertoires of behavioral skills, but also the faculties to quickly acquire new behaviors as demanded by their current situation and the needs of their users. Reinforcement learning (RL) in principle enables autonomous acquisition of such skills. However, each skill must be learned individually at considerable cost in time and effort. In this paper, we instead explore how general-purpose robotic policies can be acquired by conditioning policies on task representations. This question has previously been investigated by learning *goal-conditioned* or *universal* policies, which take in not only the current state, but also some representation of a *goal* state. However, such a task representation cannot capture many of the behaviors we might actually want a versatile robotic system to perform, since it can only represent behaviors that involve reaching individual states. For example, for a robot packing items into a box, the task is defined by the position of the items relative to the box, rather than their absolute locations in space, and therefore does not correspond to a single state configuration. How can we parameterize a more general class of behaviors, so as to make it possible to acquire truly general-purpose policies that, if conditioned appropriately, could perform any desired task?

To make it possible to learn general-purpose policies that can perform any task, we instead consider conditioning a policy on a full *distribution* over goal states. Rather than reaching a specific state, a policy must learn to reach states that have high likelihood under the provided distribution, which may specify various covariance relationships (e.g., as shown in Figure 1, that the position of the items should covary with the position of the box). In fact, we show that, because optimal policies are invariant to additive factors in reward functions, arbitrary goal distributions can represent *any* state-dependent reward function, and therefore any task. Choosing a specific distribution class provides a

*equal contribution

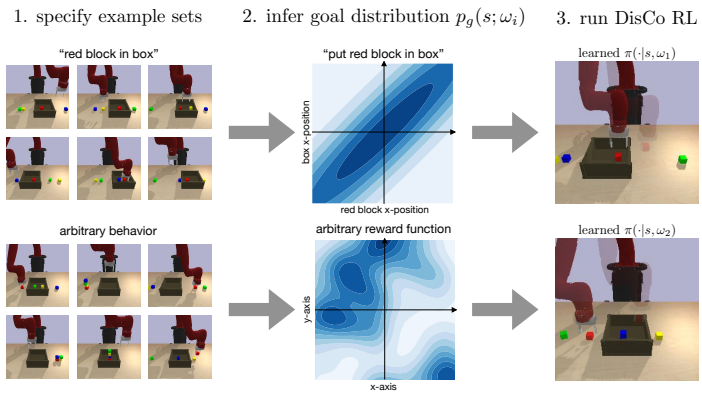


**Figure 1:** We infer the distribution parameters $\omega$ from data and pass it to a DisCo policy. Distribution-conditioned RL can express a broad range of tasks, from defining relationships between different state components (top) to more arbitrary behavior (bottom).

natural mechanism to control the expressivity of the policy. We may choose a small distribution class to narrow the range of tasks and make learning easier, or we may choose a large distribution class to expand the expressiveness of the policy.

Our experiments demonstrate that distribution-conditioned policies can be trained efficiently by sharing data from a variety of tasks and relabeling the goal distribution parameters, where each distribution corresponds to a different task reward. Lastly, while the distribution parameters can be provided manually to specify tasks, we also present two ways to infer these distribution parameters directly from data.

The main contribution of this paper is DisCo RL, an algorithm for learning distribution-conditioned policies. To learn efficiently, DisCo RL uses off-policy training and a novel distribution relabeling scheme. We evaluate on robot manipulation tasks in which a single policy must solve multiple tasks that cannot be expressed as reaching different goal states. We find that conditioning the policies on goal distributions results in significantly faster learning than solving each task individually, enabling policies to acquire a broader range of tasks than goal-conditioned methods.

## 2 Related Work

In goal-conditioned RL, a policy is given a goal state, and must take actions to reach that state [18, 43, 35, 7, 38, 30, 31, 51, 39, 17]. However, as discussed previously, many tasks cannot be specified with a single goal state. To address this, many goal-conditioned methods manually design a goal space that explicitly excludes some state variables [1, 26, 40, 13, 37, 5], for example by only specifying the desired location of an object. This requires manual effort and user insight, and does not generalize to environments with high-dimensional state representations, such as images, where manually specifying a goal space is very difficult. With images, a number of methods learn latent representations for specifying goal states [23, 31, 39, 32, 33], which makes image-based goals more tractable, but does not address the representation issues discussed above. Our work on goal distributions can be seen as a generalization of goal state reaching. Reaching a single goal state $g$ is equivalent to maximizing the likelihood of a delta-distribution centered at $g$, and ignoring some state dimensions is equivalent to maximizing the likelihood of a distribution that places uniform likelihood across the respective ignored state variables. But more generally, goal distributions capture the set of all reward functions, enabling policies to be conditioned on arbitrary tasks.

A number of prior methods learn rewards [49, 15] or policies [48, 44, 27, 11] using expert trajectories or observations. In this work, we also demonstrate that we can use observations to learn reward functions, but we have different objectives and assumptions as compared to prior work. Many of these prior methods require state sequences from expert demonstrations [48, 44, 27, 11], whereas our work only requires observations of successful outcomes to fit the goal distribution. Fu et al. [15] also only uses observations of successful outcomes to construct a reward function, but focuses on solving single tasks or goal-reaching tasks, whereas we study the more general setting where the policy is conditioned on a goal distribution.

Parametric representations of rewards have also been in the context of successor features [22, 2, 4, 3], which parameterize reward functions as linear combinations of known features. We present a general framework in which arbitrary rewards can be represented as goal distributions rather than feature weights, and also demonstrate that these goal distributions can be learned from data.

Prior work on state marginal matching [24] attempts to make a policy's stationary distribution match a target distribution to explore an environment. In our work, rather than matching a target distribution, we use the log-likelihood of a goal distributions to define a reward function, which we then maximize with standard reinforcement learning.

## 3 Background

Reinforcement learning (RL) frames reward maximization in a Markov decision process (MDP), defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, r, p, p_0, \gamma)$ [46], where $\mathcal{S}$ denotes the state space and $\mathcal{A}$ denotes the action space. In each episode, the agent's initial state $\mathbf{s}_0 \in \mathcal{S}$ is sampled from an initial state distribution $\mathbf{s}_0 \sim p_0(\mathbf{s}_0)$, the agent chooses an action $\mathbf{a} \in \mathcal{A}$ according to a stochastic policy $\mathbf{a}_t \sim \pi(\cdot|\mathbf{s}_t)$, and the next state is generated from the state transition dynamics $\mathbf{s}_{t+1} \sim p(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)$. We will use $\tau$ to denote a trajectory sequence $(\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \dots)$ and denote sampling as $\tau \sim \pi$ since we assume a fixed initial state and dynamics distribution. The objective of an agent is to maximize the sum of discounted rewards, $\mathbb{E}_{\tau\sim\pi}\left[\sum_{t=1}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right]$.

**Off-policy, temporal-difference algorithms.** Our method can be used with any off-policy temporal-difference (TD) learning algorithm. TD-learning algorithms only need $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ tuples to train a policy, where $\mathbf{a}_t$ is an action taken from state $\mathbf{s}_t$, and where $r_t$ and $\mathbf{s}_{t+1}$ are the resulting reward and next state, respectively, sampled from the environment. Importantly, these tuples can be collected by any policy, making it an off-policy algorithm. These tuples are typically sampled from a replay buffer $\mathcal{R}$, which consists of tuples generated by all previous environment interactions.

## 4 Distribution-Conditioned Reinforcement Learning

In this section, we show how conditioning policies on a goal distribution results in a MDP that can capture any set of reward functions. Each distribution represents a different reward function, and so choosing a distribution class provides a natural mechanism to choose the expressivity of the contextual policy. We then present distribution-conditioned reinforcement learning (DisCo RL), an off-policy algorithm for training policies conditioned on parametric representation of distributions, and discuss the specific representation that we use.

### 4.1 Generality of Goal Distributions

We assume that the goal distribution is in a parametric family, with parameter space $\Omega$, and augment the MDP state space with the goal distribution, as in $\mathcal{S}' = \mathcal{S} \times \Omega$. At the beginning of each episode, a parameter $\omega \in \Omega$ is sampled from some parameter distribution $p_\omega$. The parameter $\omega$ defines the goal distribution $p_g(\mathbf{s}; \omega) : \mathcal{S} \mapsto \mathbb{R}_+$ over the state space. The policy is conditioned on this parameter, and is given by $\pi(\cdot \mid \mathbf{s}, \omega)$. The objective of a distribution-conditioned (DisCo) policy is to reach states that have high log-likelihood under the goal distribution, which can be expressed as

$$\max_{\pi} \mathbb{E}_{\tau\sim\pi(\cdot|\mathbf{s},\omega)}\left[\sum_t \gamma^t \log p_g(\mathbf{s}_t; \omega)\right]. \tag{1}$$

This formulation can express arbitrarily complex distributions and tasks, as we illustrate in Figure 1. More formally:

**Remark 1** *Any reward maximization problem can be equivalently written as maximizing the log-likelihood under a goal distribution (Equation 1), up to a constant factor.*

This statement is true because, for any reward function of the form $r(\mathbf{s})$, we can define a distribution $p_g(\mathbf{s}) \propto e^{r(\mathbf{s})}$, from which we can conclude that maximizing $\log p_g(\mathbf{s})$ is equivalent to maximizing $r(\mathbf{s})$, up to the constant normalizing factor in the denominator. If the reward function depends on

the action, $r(\mathbf{s}, \mathbf{a})$, we can modify the MDP and append the previous action to the state $\bar{\mathbf{s}} = [\mathbf{s}, \mathbf{a}]$, reducing it to another MDP with a reward function of the form $r(\bar{\mathbf{s}})$.

Of course, while any reward can be expressed as the log-likelihood of a goal distribution, a specific fixed parameterization $p_g(\mathbf{s}; \omega)$ may not by itself be able to express any reward. In other words, choosing the distribution parameterization is equivalent to choosing the set of reward functions that the conditional policy can maximize. As we discuss in the next section, we can trade-off expressivity and ease of learning by choosing an appropriate goal distribution family.

### 4.2 Goal Distribution Parameterization

Different distribution classes represent different types of reward functions. To explore the different capabilities afforded by different distributions, we study three families of distributions.

**Gaussian distribution** A simple class of distributions is the family of multivariate Gaussian. Given a state space in $\mathbb{R}^n$, the distribution parameters consists of two components, $\omega = (\mu, \Sigma)$, where $\mu \in \mathbb{R}^n$ is the mean vector and $\Sigma \in \mathbb{R}^{n \times n}$ is the covariance matrix. When inferring the distribution parameters with data, we regularize the $\Sigma^{-1}$ matrix by thresholding absolute values below $\epsilon = 0.25$ to zero. With these parameters, the reward from Equation 1 is given by $r(\mathbf{s}; \omega) = -0.5(\mathbf{s} - \mu)^T \Sigma^{-1} (\mathbf{s} - \mu)$, where we have dropped constant terms that do not depend on the state $\mathbf{s}$. This simple parameterization can express a large number of reward functions. Using this parameterization, the weight of individual state dimensions depend on the values along the diagonal of the covariance matrix. By using off-diagonal covariance values, this parameterization also captures the set of tasks in which state components need to covary, such as when the one object must be placed near another one, regardless of the exact location of those objects (see the top half of Figure 1).

**Gaussian mixture model** A more expressive class of distributions that we study is the Gaussian mixture model with 4 modes, which can represent multi-modal tasks. The parameters are the mean and covariance of each Gaussian and the weight assigned to each Gaussian distribution. The reward is given by the log-likelihood of a state.

**Latent variable model** To study an even more express class of distribution, we consider a class of distributions parameterized by neural networks. Distributions parameterized by neural networks can be extremely expressive [8, 50], but distributions based on neural networks often have millions of parameters [8, 20].

To obtain an expressive yet compact parameterization, we consider non-linear reparameterizations of the original state space. Specifically, we model a distribution over the state space using a Gaussian variational auto-encoder (VAE) [19, 42]. Gaussian VAEs model a set of observations using a latent-variable model of the form $p(\mathbf{s}) = \int_{\mathcal{Z}} p(\mathbf{z}) p_{\psi_d}(\mathbf{s} \mid \mathbf{z}) d\mathbf{z}$, where $\mathbf{z} \in \mathcal{Z} = \mathbb{R}^{d_z}$ are latent variables with dimension $d_z$. The distribution $p_{\psi_d}$ is a learned generative model or "decoder" and $p(z)$ is a standard multivariate Gaussian distribution in $\mathbb{R}^{d_z}$. A Gaussian VAE also learns a posterior distribution or "encoder" that maps states $\mathbf{s}$ onto Gaussian distributions in a latent space, given by $q_{\psi_e}(\mathbf{z}; \mathbf{s})$. We refer readers to Doersch [9] for a detailed explanation of VAEs.

Gaussian VAEs are explicitly trained so that Gaussian distributions in a latent space define distributions over the state space. Therefore, we represent a distribution over the state space with the mean $\mu_z \in \mathbb{R}^{d_z}$ and variances $\sigma_z \in \mathbb{R}^{d_z}$ of a diagonal Gaussian distribution *in the learned, latent space*, which we write as $\mathcal{N}(\mathbf{z}; \mu_z, \sigma_z)$. In other words, the parameters $\omega = (\mu_z, \sigma_z)$ define the following distribution over the states:

$$p_g(\mathbf{s}; \omega) = \int_{\mathcal{Z}} \mathcal{N}(\mathbf{z}; \mu_z, \sigma_z) p_{\psi_d}(\mathbf{s} \mid \mathbf{z}) d\mathbf{z}, \tag{2}$$

where $p_{\psi_d}(\mathbf{s} \mid \mathbf{z})$ is the generative model from the VAE. Because $\mathbf{z}$ resides in a learned latent space, the set of reward functions that can be expressed with Equation (2) includes arbitrary non-linear transformations of $\mathbf{s}$.

## 5 Learning Goal Distributions and Policies

Given any of the distribution classes mentioned above, we now consider how to obtain a specific goal distribution parameter $\omega$ and train a policy to maximize Equation (1). We start with obtaining goal distribution parameters $\omega$. While a user can manually select the goal distribution parameters $\omega$, this requires a degree of user insight, which can be costly or practically impossible if using the latent representation in Equation (2). We discuss two automated alternatives for obtaining the goal distribution parameters $\omega$.

### 5.1 Inferring Distributions from Examples

One simple and practical way to specify a goal distribution is to provide $K$ example observations $\{\mathbf{s}_k\}_{k=1}^K$ in which the task is successfully completed. This supervision can be easier to provide than full demonstrations, which not only specify the task but also must show how to solve the task through a sequence of states $(\mathbf{s}_1, \mathbf{s}_2, \dots)$ or states and actions $(\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \dots)$. Given the example observations, we describe a way to infer the goal parameter based on the parameterization.

If $\omega$ represents the parameters of a distribution in the state space, we learn a goal distribution via maximum likelihood estimation (MLE), as in

$$\omega^* = \arg\max_{\omega \in \Omega} \sum_{k=1}^{K} \log p_g(\mathbf{s}_k; \omega), \tag{3}$$

and condition the policy on the resulting parameter $\omega^*$.

If $\omega$ represents the parameters of a distribution in the latent space, we need a Gaussian distribution in the latent space that places high likelihood on all of the states in $\mathcal{D}_{\text{subtask}}$. We obtain such a distribution by finding a latent distribution that minimizes the KL divergence to the mixture of posteriors $\frac{1}{K}\sum_k q_{\psi_e}(\omega; \mathbf{s}_k)$. Specifically, we solve the problem

$$\omega^* = \arg\min_{\omega \in \Omega} D_{\text{KL}}\left( \frac{1}{K} \sum_k q_{\psi_e}(\mathbf{z}; \mathbf{s}_k) \,\middle\|\, p(\mathbf{z}; \omega) \right), \tag{4}$$

where $\Omega$ is the set of all means and diagonal covariance matrices in $\mathbb{R}^{d_z}$. The solution to Equation (4) can be computed in closed form using moment matching [28].

### 5.2 Dynamically Generating Distributions via Conditioning

We also study a different use case for training a DisCo policy: automatically decomposing long-horizon tasks into sub-tasks. Many complex tasks lend themselves to such a decomposition, and learning each of these sub-tasks can be significantly faster than directly solving the main task. For example, a robot that must arrange a table can divide this task into placing one object to a desired location before moving on to the next object. To automatically decompose a task, we need to convert a "final task," represented as $\mathcal{T}$, into $M$ different sub-tasks, where $M$ is the number of sub-tasks needed to accomplish the final task. Moreover, rather than training a separate policy for each sub-task, we would like to train a single policy that can accomplish all of these sub-tasks. How can we convert a task parameter $\mathcal{T}$ into different sub-tasks, all of which can be accomplished by a single policy?

We can address this question by training a DisCo policy. A task represented by parameters $\mathcal{T}$ can be decomposed into a sequence of sub-tasks, each of which in turn is represented by a goal distribution. We accomplish this by learning *conditional distributions*, $p_g^i(\mathbf{s} \mid \mathcal{T})$: conditioned on some final task $\mathcal{T}$, the conditional distribution $p_g^i(\mathbf{s} \mid \mathcal{T})$ is a distribution over desired states for sub-task $i$.

To obtain a conditional distribution $p_g^i(\mathbf{s} \mid \mathcal{T})$ for sub-task $i$, we assume

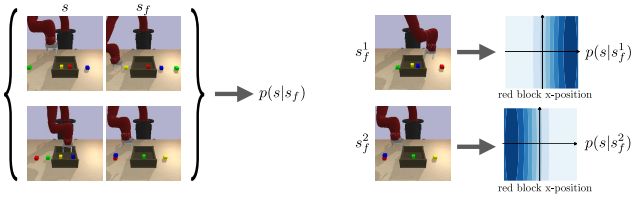


**Figure 2:** A robot must arrange objects into a configuration $\mathbf{s}_f$ that changes from episode to episode. This task consists of multiple sub-tasks, such as first moving the red object to the correct location. Given a final state $\mathbf{s}_f$, there exists a distribution of intermediate states $\mathbf{s}$ in which the first sub-task is completed. We use pairs of states $\mathbf{s}, \mathbf{s}_f$ to learn a conditional distribution that defines the first sub-task given the final task, $p(\mathbf{s} \mid \mathbf{s}_f)$.

access to tuples $\mathcal{D}^i_{\text{subtask}} = \{(\mathbf{s}^{(k)}, \mathcal{T}^{(k)})\}_{k=1}^K$, where $\mathbf{s}^{(k)}$ is an example state in which sub-task $i$ is accomplished for solving task $\mathcal{T}^{(k)}$. See Figure 2 for a visualization. Our experiments test the setting where final tasks are represented by a final state $\mathbf{s}_f$ that we want the robot to reach, meaning that $\mathcal{T} = \mathbf{s}_f$. We note that one can train a goal-conditioned policy to reach this final state $\mathbf{s}_f$ directly, but, as we will discuss in Section 6, this decomposition significantly accelerates learning by exploiting access to the pairs of states.

For each $i$, and dropping the dependence on $i$ for clarify, we learn a conditional distribution, by fitting a joint Gaussian distribution, denoted by $p_{\mathbf{S},\mathbf{S}_f}(\mathbf{s}, \mathbf{s}_f; \mu, \Sigma)$, to these pairs of states using MLE, as in

$$\mu^*, \Sigma^* = \arg\max_{\mu,\Sigma} \sum_{k=1}^{K} \log p_{\mathbf{S},\mathbf{S}_f}(\mathbf{s}^{(k)}, \mathbf{s}_f^{(k)}; \mu, \Sigma). \tag{5}$$

This procedure requires up-front supervision for each sub-task during training time, but at test time, given a desired final state $\mathbf{s}_f$, we compute the parameters of the Gaussian conditional distribution $p_g(\mathbf{s} \mid \mathbf{s}_f; \mu^*, \Sigma^*)$ in closed form. Specifically, the conditional parameters are given by

$$\begin{aligned} \bar{\mu} &= \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(\mathbf{s}_f - \mu_2) \\ \bar{\Sigma} &= \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}, \end{aligned} \tag{6}$$

where $\mu_1$ and $\mu_2$ represents the first and second half of $\mu^*$, and similarly for the covariance terms. To summarize, we learn $\mu^*$ and $\Sigma^*$ with Equation (5) and then use Equation (6) to automatically transform a final task $\mathcal{T} = \mathbf{s}_f$ into a goal distribution $\omega = (\bar{\mu}, \bar{\Sigma})$ which we give to a DisCo policy. Having presented two ways to obtain distributions, we now turn to learning DisCo RL policies.

### 5.3 Learning Distribution-Conditioned Policies

In this section, we discuss how to optimize Equation 1 using an off-policy TD algorithm. As discussed in Section 3, TD algorithms require tuples of state, action, next state, and reward, denoted by $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$. To collect this data, we condition a policy on a goal distribution parameter $\omega$, collect a trajectory with the policy $\tau = [\mathbf{s}_0, \mathbf{a}_0, \cdots]$, and then store the trajectory and distribution parameter into a replay buffer [29], denoted as $\mathcal{R}$. We then sample data from this replay buffer to train our policy using an off-policy TD algorithm. We use soft actor-critic as our RL algorithm [16], though in theory any off-policy algorithm could be used.

Because TD algorithms are off-policy, we propose to reuse data collected by a policy conditioned on one goal distribution $\omega$ to learn about how a policy should behave under another goal distribution $\omega'$. In particular, given a state $\mathbf{s}$ sampled from a policy that was conditioned on some goal distribution parameters $\omega$, we occasionally relabel the goal distribution with an alternative goal distribution $\omega' = RS(\mathbf{s}, \omega)$ for training, where $RS$ is some relabeling strategy. For relabeling, given a state $\mathbf{s}$ and existing parameters $\omega = (\mu, \Sigma)$, we would like to provide a strong learning signal by creating a distribution parameter that gives high reward to an achieved state. We used a simple strategy that we found worked well: we replace the mean with the state vector $\mathbf{s}$ and randomly re-sampling the covariance from the set of observed covariances as in $RS(\mathbf{s}, (\mu, \Sigma)) = (\mathbf{s}, \Sigma')$, where $\Sigma'$ is sampled uniformly from the replay buffer. This relabeling is similar to relabeling methods used in goal-conditioned reinforcement learning [18, 1, 40, 38, 30, 37, 31, 12], but applied to goal distributions rather than individual goal states.

We call the method *DisCo RL* when learning a distribution from examples and *Conditional DisCo RL* when learning a conditional distribution. We summarize both in Algorithm 1.

## 6 Experiments

Our experiments study the following questions: **(1)** How does DisCo RL with a learned distribution compare to prior work that also uses successful states for computing rewards? **(2)** Can we apply Conditional DisCo RL to solve long-horizon tasks that are decomposed into shorter sub-tasks? **(3)** How do DisCo policies perform when conditioned on goal distributions that were never used for data-collection? The first two questions evaluate the variants of DisCo RL presented in Section 5, and the third question studies how well the method can generalize to test time task specifications. We also include ablations that study the importance of the relabeling strategy presented in Section 5.3.

**Algorithm 1** (Conditional) Distribution-Conditioned RL

**Require:** Policy $\pi_\theta$, $Q$-function $Q_\phi$, TD algorithm $\mathcal{A}$, relabeling strategy $RS$, exploration parameter distribution $p_\omega$, replay buffer $\mathcal{R}$.

Compute $\omega$ using Equation (4) or (5) (if unconditional).
**for** $0, \ldots, N_{\text{episode}} - 1$ episodes **do**
  Sample $\mathbf{s}_f$ and compute $\omega$ with Equation (6).
  Sample trajectory from environment $\tau \sim \pi(\cdot|\mathbf{s}; \omega)$ and store tuple $(\tau, \omega)$ in replay buffer $\mathcal{R}$.
  **for** $0, \ldots, N_{\text{updates}} - 1$ steps **do**
    Sample trajectory and parameter $(\tau, \omega) \sim \mathcal{R}$.
    Sample transition tuple $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ and a future state $\mathbf{s}_h$ from $\tau$, where $t < h$.
    With probability $p_{\text{relabel}}$, relabel $\omega \leftarrow RS(\mathbf{s}_h, \omega)$.
    Compute reward $r = \log p_g(\mathbf{s}_t; \omega)$ and augment states $\hat{\mathbf{s}}_t \leftarrow [\mathbf{s}_t; \omega]$, $\hat{\mathbf{s}}_{t+1} \leftarrow [\mathbf{s}_{t+1}; \omega]$.
    Update $Q_\phi$ and $\pi_\theta$ using $\mathcal{A}$ and $(\hat{\mathbf{s}}_t; \mathbf{a}_t, \hat{\mathbf{s}}_{t+1}, r)$.

We study these questions in three simulated manipulation environments of varying complexity, shown in Figure 3. We first consider a simple two-dimensional "Flat World" environment in which an agent can pick up and place objects at various locations. The second environment contains a Sawyer robot, a rectangular tray, and four blocks, which the robot must learn to manipulate. The agent controls the velocity of the end effector and gripper, and the arm is restricted to move in a 2D plane perpendicular to the table's surface. Lastly, we use an IKEA furniture assembly environment from Lee et al. [25]. An agent controls the velocity of a cursor that can lift and place 3 shelves onto a pole. Shelves are connected automatically when they are within a certain distance of the cursor or pole. The states comprise the Cartesian position of all relevant objects and, for the Sawyer task, the gripper state. For the Sawyer environment, we also consider an image-based version which uses a 48x48 RGB image for the state. All plots show mean and standard deviation across 5 seeds, as well as optimal performance (if non-zero) with a dashed line.

### 6.1 Learning from Examples

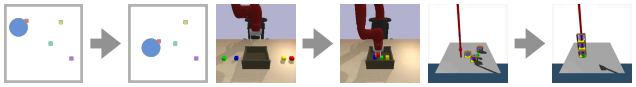

**Figure 3:** Illustrations of the experimental domains, in which a policy must (left) use the blue cursor to move objects to different locations, (center) control a Sawyer arm to move cubes into and out of a tray, and (right) attach shelves to a pole using a cursor.

Our first set of experiments evaluates how well DisCo RL performs when learning distribution parameters from a fixed set of examples, as described in Section 5.1. We test all three parameterizations from Section 4.2 on different environments: First, we evaluate DisCo RL with a Gaussian model learned via Equation (3) on the Sawyer environment, in which the policy must move the red object into the bowl and ignore three "distractor" objects. Second, we evaluate DisCo RL with a GMM model learned via Equation (3) on the Flat World environment, where the agent must move a specific object to any one of four locations. Lastly, we evaluate DisCo RL with a latent variable model learned via Equation (4) on an image-based version of the Sawyer environment, where the policy must move the hand to a fixed location and ignore visual distractions. This last experiment is done completely from images, where manually specifying the parameters of a Gaussian in image-space would be impractical. The experiments used between $K = 30$ to 50 examples for learning the goal distribution parameters. We report the normalized final distance, where we normalize by the final distance achieved by a random policy.

We compare to past work that uses example states to learn a reward function. Specifically, we compare to variational inverse control with events (**VICE**) [15], which trains a success classifier to predict the user-provided example states as positive and replay buffer states as negative, and then uses the log-likelihood of the classifier as a reward. We also include an oracle labeled **SAC (oracle reward)** which uses the ground truth reward. Since VICE requires training a separate classifier for each task, these experiments only test the methods on a single task. Note that a single DisCo RL policy can solve multiple tasks by conditioning on different distribution parameters, as we will study in the next section.

We see in Figure 4 that DisCo RL often matches the performance of using an oracle reward and consistently outperforms VICE. VICE often failed to learn, possibly because the method was developed using hundreds to thousands of examples, where as we only provided 30 to 50 examples.

### 6.2 Conditional Distributions for Sub-Task Decomposition

The next experiments study how Conditional DisCo RL can automatically decompose a complex task into easier sub-tasks. We design tasks that require reaching a desired final state $\mathbf{s}_f$, but that can be decomposed into smaller sub-tasks. The first task requires controlling the Sawyer robot to move $4$ blocks with randomly initialized positions into a tray at a fixed location. We design analogous tasks in the IKEA environment (with $3$ shelves and a pole) and Flat World environment (with $4$ objects). All of these tasks can be split into sub-task that involving moving a single object at a time.

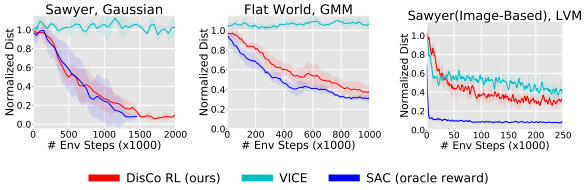


**Figure 4:** (Lower is better.) Learning curve showing normalized distance versus environment steps of various method. DisCo RL uses a (left) Gaussian model, (middle) Gaussian mixture model, or (right) latent-variant model on their respective tasks. DisCo RL with a learned goal distribution consistently outperforms VICE and obtains a final performance similar to using oracle rewards.

For each object $i = 1, \dots, M$, we collect an example set $\mathcal{D}^i_{\text{subtask}}$. As described in Section 5.2, each set $\mathcal{D}^i_{\text{subtask}}$ contains $K = 30$ to $50$ pairs of state $(\mathbf{s}, \mathbf{s}_f)$, in which object $i$ is in the same location as in the final desired state $\mathbf{s}_f$, as shown in Figure 2. We fit a joint Gaussian to these pairs using Equation (5). During exploration and evaluation, we sample an initial state $\mathbf{s}_0$ and final goal state $\mathbf{s}_f$ uniformly from the set of possible states, and condition the policy on $\omega$ given by Equation (6). Because the tasks are randomly sampled, this setting also tested the ability for Conditional DisCo RL to generalize to new goal distributions.

We evaluate how well DisCo RL can solve long-horizon tasks by conditioning the policy on each $p^i_g(\mathbf{s} \mid \mathbf{s}_f)$ sequentially for $H/M$ time steps. We report the cumulative number of tasks that were solved, where each task is considered solved when the respective object is within a minimum distance of its target location specified by $\mathbf{s}_f$. For the IKEA environment, we also consider moving the pole to the correct location as a task. We compare to VICE which trains separate classifiers and policies for each sub-task using the examples and also sequentially runs each policies for $H/M$ time steps.

One can train a goal-conditioned policy to reach this final state $\mathbf{s}_f$ directly, and so we compare to hindsight experience replay (**HER**) [1], which attempts to directly reach the final state for $H$ time steps and learns using an oracle dense reward. We see in Figure 5 that Conditional DisCo RL significantly outperforms VICE and HER and that Conditional DisCo RL successfully generalizes to new goal distributions.

**Ablations** Lastly, we include ablations that test the importance of relabeling the mean and covariance parameters during training. We see in Figure 5 that relabeling both parameters, and particularly the mean, accelerates learning.

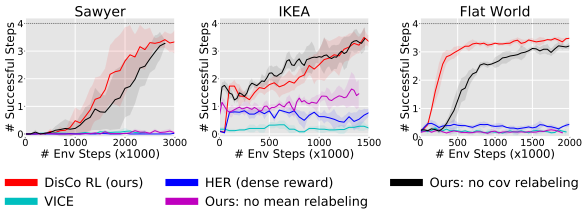


**Figure 5:** (Higher is better.) Learning curves showing the number of cumulative tasks completed versus environment steps for the Sawyer (left), IKEA (middle), and Flat World (right) tasks. We see that DisCo RL significantly outperforms HER and VICE, and that relabeling the mean and covariance is important.

## 7 Discussion

We presented DisCo RL, a method for learning general-purpose policies specified using a goal distribution. Our experiments show that DisCo policies can solve a variety of tasks using goal distributions inferred from data, and can accomplish tasks specified by goal distributions that were not seen during training. In this work, we studied Gaussian, Gaussian mixture, and latent variable parameterizations of goal distributions using an existing dataset. An exciting direction for future work would be to interleave DisCo RL and distribution learning, for example by using newly acquired data to update the learned VAE or by backpropagating the DisCo RL loss into the VAE learning loss. Another promising direction would be to study goal-distribution directed exploration, in which an agent can explore along certain distribution of states, analogous to work on goal-directed exploration [21, 10, 41, 39].

## References


[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob Mcgrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. In *NeurIPS*, 2017. URL `https://arxiv.org/pdf/1707.01495.pdfhttp://arxiv.org/abs/1707.01495`.

[2] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In *NeurIPS*, pages 4055–4065, 2017.

[3] Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *ICML*, pages 501–510, 2018.

[4] Diana Borsa, Andre Barreto, John Quan, Daniel J Mankowitz, Hado van Hasselt, Remi Munos, David Silver, and Tom Schaul. Universal successor features approximators. In *ICLR*, 2018.

[5] Cédric Colas, Pierre Fournier, Mohamed Chetouani, Olivier Sigaud, and Pierre-Yves Oudeyer. Curious: intrinsically motivated modular multi-goal reinforcement learning. In *ICML*, pages 1331–1340, 2019.

[6] Erwin Coumans et al. Bullet physics library. *Open source: bulletphysics. org*, 15(49):5, 2013.

[7] Vikas Dhiman, Shurjo Banerjee, Jeffrey M Siskind, and Jason J Corso. Floyd-warshall reinforcement learning: Learning from past experiences to reach new goals. *arXiv:1809.09318*, 2018.

[8] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv:1605.08803*, 2016.

[9] Carl Doersch. Tutorial on variational autoencoders. *arXiv:1606.05908*, 2016.

[10] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *CoRR*, abs/1901.10995, 2019.

[11] Ashley Edwards, Himanshu Sahni, Yannick Schroecker, and Charles Isbell. Imitating latent policies from observation. In *ICML*, pages 1755–1763, 2019.

[12] Benjamin Eysenbach, Xinyang Geng, Sergey Levine, and Ruslan Salakhutdinov. Rewriting history with inverse rl: Hindsight inference for policy improvement.

[13] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *ICML*, 2018.

[14] Justin Fu, Katie Luo, and Sergey Levine. Learning Robust Rewards With Adversarial Inverse Reinforcement Learning. In *ICLR*, 2018. URL `https://arxiv.org/pdf/1710.11248.pdf`.

[15] Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. Variational inverse control with events: A general framework for data-driven reward definition. In *NeurIPS*, pages 8538–8547, 2018.

[16] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *ICML*, 2018. URL `https://arxiv.org/pdf/1801.01290.pdf`.

[17] Tom Jurgenson, Edward Groshev, and Aviv Tamar. Sub-goal trees–a framework for goal-directed trajectory prediction and optimization. *arXiv:1906.05329*, 2019.

[18] L P Kaelbling. Learning to achieve goals. In *IJCAI*, volume vol.2, pages 1094 – 8, 1993.

[19] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. In *ICLR*, 2014. URL `https://arxiv.org/pdf/1312.6114.pdf`.

[20] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in neural information processing systems*, pages 10215–10224, 2018.

[21] Sven Koenig and Reid G Simmons. The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning*, 22(1-3):227–250, 1996.

[22] Tejas D Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman. Deep successor reinforcement learning. *arXiv:1606.02396*, 2016.

[23] Adrien Laversanne-Finot, Alexandre Pere, and Pierre-Yves Oudeyer. Curiosity driven exploration of learned disentangled goal spaces. In *CoRL*, pages 487–504, 2018.

[24] Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. In *ICLR*, 2019.

[25] Youngwoon Lee, Edward S Hu, Zhengyu Yang, Alex Yin, and Joseph J Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. *arXiv:1911.07246*, 2019. URL `https://clvrai.com/furniture`.

[26] Andrew Levy, Robert Platt, and Kate Saenko. Hierarchical Actor-Critic. *arXiv:1712.00948*, 2017.

[27] Yuxuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from Observation: Learning to Imitate Behaviors from Raw Video via Context Translation. In *ICRA*, 2018. URL `https://arxiv.org/pdf/1707.03374.pdf`.

[28] Thomas P Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369, 2001.

[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and Others. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[30] Ofir Nachum, Google Brain, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-Efficient Hierarchical Reinforcement Learning. In *NeurIPS*, 2018. URL `https://sites.google.com/view/efficient-hrl`.

[31] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual Reinforcement Learning with Imagined Goals. In *NeurIPS*, 2018. URL `https://sites.google.com/site/`.

[32] Ashvin Nair, Shikhar Bahl, Alexander Khazatsky, Vitchyr Pong, Glen Berseth, and Sergey Levine. Contextual imagined goals for self-supervised robotic learning. In *CoRL*, pages 530–539, 2020.

[33] Suraj Nair, Silvio Savarese, and Chelsea Finn. Goal-aware prediction: Learning to model what matters. In *ICML*, 2020.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[35] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell. Zero-Shot Visual Imitation. In *ICLR*, 2018. URL `https://arxiv.org/pdf/1804.08606.pdf`.

[36] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *RSS*, 2020.

[37] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob Mcgrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research. *arXiv:1802.09464*, 2018. URL `http://fetchrobotics.com/`.

[38] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal Difference Models: Model-Free Deep RL For Model-Based Control. In *ICLR*, 2018. URL `https://arxiv.org/pdf/1802.09081.pdf`.

[39] Vitchyr H. Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *CoRR*, abs/1903.03698, 2019.

[40] Paulo Rauber, Filipe Mutz, and Juergen Jürgen Schmidhuber. Hindsight policy gradients. In *CoRR*, volume abs/1711.0, 2017. URL `https://arxiv.org/pdf/1711.06006.pdf`.

[41] Zhizhou Ren, Kefan Dong, Yuan Zhou, Qiang Liu, and Jian Peng. Exploration via hindsight goal generation. In *NeurIPS*, pages 13485–13496, 2019.

[42] Danilo J Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *ICML*, 2014. URL https://arxiv.org/pdf/1401.4082.pdf.

[43] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal Value Function Approximators. In *ICML*, 2015.

[44] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *ICRA*, 2018.

[45] Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. End-to-End Robotic Reinforcement Learning without Reward Engineering. In *RSS*, 2019. URL http://arxiv.org/abs/1904.07854.

[46] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. 1998. URL http://incompleteideas.net/sutton/book/bookdraft2016sep.pdfhttps://webdocs.cs.ualberta.ca/{~}sutton/book/bookdraft2016sep.pdf.

[47] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, Vincent Vanhoucke, Google Brain, and Google Deepmind. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. In *RSS*, 2018. URL https://arxiv.org/pdf/1804.10332.pdf.

[48] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In *IJCAI*, 2018.

[49] Faraz Torabi, Garrett Warnell, and Peter Stone. Generative adversarial imitation from observation. *arXiv:1807.06158*, 2018.

[50] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016.

[51] David Warde-Farley, Tom Van De Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Mnih Volodymyr. Unsupervised Control Through Non-Parametric Discriminative Rewards. In *ICLR*, 2019.

[52] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond Empirical Risk Minimization. In *ICLR*, oct 2018. URL http://arxiv.org/abs/1710.09412.

## A Generalization to Real-World Robots

While our experiments were only conducted in simulated environments, we expect DisCo RL to generalize to real-world robots. We note that both goal-conditioned RL [31] and VICE [45] have been applied to real-world robot domains, and that Figure 5 demonstrates that DisCo RL can significantly outperform these methods in multi-task settings where tasks are not specified by single goals. We note that these findings are consistent across all three simulated domains, suggesting that these results are general properties of the methods and may apply to other domains, including real-world robotics. We also note that the PyBullet simulator [6] has been successfully applied for sim-to-real transfer [47, 36], suggesting that strong performance in the simulator can generalize to real-world robots.

Lastly, in our experiments DisCo RL used $30$ to $50$ examples to learn goal distributions that specify different robot tasks. Specifying such few number of example successful states is particularly practical for real-world domains, where specifying tasks often requires manually specifying reward functions or adding task-specific instrumentation and sensors.

## B Environments

**Sawyer** This environment is based in the PyBullet [6] physics simulator. It consists of a Sawyer robot mounted next to a table, on top of which there is a tray and four blocks. The robot must learn to manipulate the blocks via its gripper. The robot is controlled via position control, and it is restricted to move in a 2D plane. Specifically, the robot arm can move in the YZ coordinate plane and the gripper can open along the X axis, where the X, Y, Z axes move along the front-back, left-right, and up-down directions of table, respectively. We also constrain the objects to move along the YZ coordinate plane. The agent has access to state information, comprising of the position of the end effector and gripper state, as well as the positions of the objects and tray.

**Visual Sawyer** We use the same environment as described above for the vision-based tasks but replace the state with 48x48 RGB images. We pretrain a VAE on 5120 randomly generated images and obtain the goal distribution using 30 example images. We visualize some example images in Figure 6.

The encoder consist of a 3 convolutions with the following parameters

1. channels: 64, 128, 128
2. kernel size: 4, 4, 3
3. stride: 2, 2, 2
4. padding: 1, 1, 1

followed by 3 residual layers each containing two convolutions with the following parameters

1. channels: 64, 64
2. kernel size: 3, 1
3. stride: 1, 1
4. padding: 1, 1

and two linear layers that projects the convolution output into the mean and log-standard deviation of a Gaussian distribution in a latent dimension with dimension 64.

The decoder begins with a linear layer with the transposed shape as the final encoder linear layer, followed by a reshaping into a latent image the same shape as the final encoder convolution output shape. This latent image is put through a convolution (128 channels, kernel size 3, stride 1, padding 1), and then an equivalent residual stack as the encoder, and two transposed convolutions with parameters

1. channels: 64, 64
2. kernel size: 4, 4
3. stride: 2, 1
4. padding: 2, 1

which outputs the mean of a Gaussian distribution with a fixed unit variance. We trained this VAE with Adam with a learn rate of $10^{-3}$ and default PyTorch [34] parameters ($\beta_1 = 0.9$ and $\beta_2 = 0.999$) for 100 epochs and annealed the loss on the KL term from 0 to 1 for the first 20 epochs.

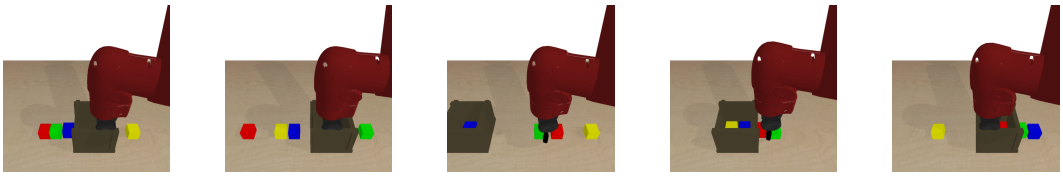

**Figure 6:** Example images with the hand in a fixed position used to obtain a goal distribution.

**IKEA** We adapted this environment from the suite of furniture assembly environments developed by Lee et al. [25]. In our environment, the agent must learn to attach a set of 3 shelves to a pole. It can do so by controlling two end effectors: one end effector that can move the pole, and another end effector that can move the shelves. The former end effector is always attached to the pole, while the latter end effector can selectively attach and detach itself from the shelves. Both end effectors can move via 3D position control, in a $1 \times 1 \times 1$ area for a maximum of $0.05$ units (in each direction) per timestep. The end effector interfacing with the shelves can hold onto a shelf by applying a grasp action when it is within the bounding box region of a shelf. Each shelf has a connection point at which point it will attach to the pole, and the pole has a receiving connection point as well. When these two connection points are within $0.2$ units away from one another, the shelf automatically attaches to the pole and becomes welded. The objects have 3 degrees of freedom via translations in 3D space. The objects are not allowed to collide with one another – if an action causes them to collide, that action is ignored by the environment and the next state is the same as the current state. As an exception, when an end effector is not grabbing an object, it is allowed to move through objects. The agent has access to state information, comprising the 3D position of the end effectors, shelves, and the pole, and indicator information for whether each end effector is grasping an object.

**Flat World** This two-dimensional environment consists of a policy and 4 objects, each of which are defined by their XY-coordinate. The policy and objects are in an enclosed $8 \times 8$ unit space. The policy's action space is three dimensional: two correspond to relative change in position, for a maximum of 1 unit in each dimension per timestep, and one corresponds to a grab action. The grab action takes on a value between $-1$ and $1$. If this grab action is positive, then the policy picks up the closest object that is within 1 unit of it, or none if there are no such objects. If this grab action is non-positive, then the policy drops any object that it was holding. While an object is grabbed, the object moves rigidly with the policy. The policy can only grab one object at a time, with ties broken by a predetermined, fixed order. The agent has access to state information, comprising the 2D position of the policy and the 4 objects.

## C Experimental Details

**Learning from Examples** These experiments were evaluated on the Sawyer and FlatWorld environment. In this section we describe the setup for the Sawyer environment with Gaussian goal distributions. In this setting, the agent needs to pick up one of the blocks (specifically, the red block) and place it into the tray. The initial position of the tray and objects vary in each episode. The objective is only to minimize the relative distance between the red block and the tray, and it is important for the agent to ignore the absolute position of the other blocks and the tray. The robot can attempt to slide the tray to a specific goal location, but the tray is heavy and moves very slowly. If it successfully moves the tray to another location, it will not have enough time in the episode to move the red block.

We generated $K = 30$ examples of successful goal states, in which the red block is always inside the tray, and the tray, other objects, hand, and gripper are in random locations and configurations. We provided this set of example goal states as input to the competing baselines. Each baseline used the example states in the following manner:

- DisCo RL: infer a goal distribution from the example states
- VICE: a classifier is trained to predict whether a state is optimal, with the example states as the positive examples for the classifier

For evaluation, a trajectory is successful if the red block is placed in the tray. We plot this success metric over time in Figure 4.

**Conditional Distributions for Sub-Task Decomposition** For the multi-task evaluations, we performed experiments in all three of our environments. For each environment, we split the task into several subtasks, as described below:

- Sawyer: move the blocks to their goal locations. Each subtask represents moving one block at a time to its goal location.
- IKEA: move the pole and the shelves to their goal locations. Each subtask represents moving the pole and one of the shelves to their goal locations.
- Flat World: move the objects to their goal locations. Each subtask represents moving one object at a time to its goal location.

We generated an example dataset of successful states for each subtask. For each example state for a particular subtask, we also provided a state representing the final configuration for the entire task (after all subtasks are solved). See Section D.2 for additional details regarding the example sets. The evaluation metrics for each environment are as follows:

- Sawyer: the number of objects that are within $0.10$ units of their respective final goal locations
- IKEA: the number of shelves that are connected to the pole, in addition to an indicator for whether the pole is within $0.10$ units of its final goal location
- Flat World: the number of objects that are within $1$ unit of their respective final goal locations

For evaluation, we provide the HER baseline oracle goals. In this setting, the provided goal is identical to the initial state at the beginning of the episode, except for the position of the red block, which we set to be inside the tray. For example, if the state is given by

$$\mathbf{s}_0 = \left[x_0^{\text{EE}}, y_0^{\text{EE}}, x_0^{\text{red-block}}, y_0^{\text{red-block}}, x_0^{\text{blue-block}}, y_0^{\text{blue-block}}, \ldots\right],$$

and we want the red block to move to a position $(x^*, y^*)$, we set the goal to

$$\mathbf{s}_g = \left[x_0^{\text{EE}}, y_0^{\text{EE}}, x^*, y^*, x_0^{\text{blue-block}}, y_0^{\text{blue-block}}, \ldots\right].$$

In theory, this oracle goal encourages the robot to focus on moving the red block to its goal rather than moving the other state components. The HER baseline only uses this oracle during evaluation.

Details regarding the distribution of final goal states used for exploration rollouts and relabeling during training, are provided in Table 1. For evaluation, we used $H = 100$ for the IKEA and Flat World environments, and $H = 400$ for the Sawyer environment.

| **Goal Use Case** | **Sawyer** | **IKEA** | **Flat World** |
|---|---|---|---|
| Exploration | $50\%$: objects on ground, $50\%$: objects in tray | Shelves assembled, pole in random position | Objects in random positions |
| Training | Same as above | Objects in random positions | Same as above |

**Table 1:** Environment specific final goal distributions.

# D Implementation Details

## D.1 General Training Algorithm and Hyperparameters

In our experiments, we use soft actor-critic as our RL algorithm [16]. For specific details on the hyperparameters that we used, see Table 3.

| **Hyper-parameter** | **Sawyer** | **IKEA** | **Flat World** |
|---|---|---|---|
| Number of examples (per subtask) $K$ | 30 | 50 | 30 |
| Std. dev. of Gaussian noise added to example set data | 0.01 | 0.1 | 0.01 |

**Table 2:** Environment specific hyper-parameters.

| Hyper-parameter | Value |
|---|---|
| horizon $H$ (for training) | 100 |
| batch size | 2048 |
| discount factor | 0.99 |
| Q-function and policy hidden sizes | $[400, 300]$ |
| Q-function and policy hidden activations | ReLU |
| replay buffer size | 1 million |
| hindsight relabeling probability | 80% |
| target network update speed $\tau$ | 0.001 |
| number of training updates per episode $N_{\text{updates per episode}}$ | 100 |
| number of training batches per environment step | 1 |

**Table 3:** General hyper-parameters used for all experiments.

### D.2 DisCo RL

**Covariance Smoothing** We apply pre-preprocessing and post-processing steps to obtain the distribution parameters used in RL. In the pre-processing phase, we add i.i.d. Gaussian noise to the dataset of examples. The amount of noise that we apply varies by environment – see Table 2 for specific details. After inferring the raw parameters of the Gaussian distribution $\mu$ and $\Sigma$, we apply post-processing steps to the covariance matrix. We begin by inverting the covariance matrix $\Sigma$. For numerical stability, we ensure that the condition number of $\Sigma$ does not exceed 100 by adding a scaled version of the identity matrix to $\Sigma$. After obtaining $\Sigma^{-1}$, we normalize its components such that the largest absolute value entry of the matrix is 1. Finally, we apply a regularization operation that thresholds all values of the matrix whose absolute value is below 0.25 to 0. We found this regularization operation to be helpful when the number of examples provided is low, to prevent the Gaussian model from inferring spurious dependencies in the data. We used the resulting $\mu$ and $\Sigma^{-1}$ for computing the reward.

**Conditional distribution details** To obtain the conditional distribution used for relabeling and multi stage planning, we assume that data is given in the form of pairs of states $\{(\mathbf{s}^{(k)}, \mathbf{s}_f^{(k)})\}_{k=1}^K$, in which $\mathbf{s}^{(k)}$ correspond to a state where a sub-task is accomplished when trying to reach the final state $\mathbf{s}_f$. We fit a joint Gaussian distribution of the form

$$p_{\mathbf{S},\mathbf{S}_f} = \mathcal{N}\left(\begin{bmatrix}\mu_{\mathbf{s}} \\ \mu_{\mathbf{s}_f}\end{bmatrix}, \begin{bmatrix}\Sigma_{\mathbf{ss}} & \Sigma_{\mathbf{ss}_f} \\ \Sigma_{\mathbf{s}_f\mathbf{s}} & \Sigma_{\mathbf{s}_f\mathbf{s}_f}\end{bmatrix}\right), \mu_{(\cdot)} \in \mathbb{R}^{\dim(\mathcal{S})}, \Sigma_{(\cdot)} \in \mathbb{R}^{\dim(\mathcal{S})\times\dim(\mathcal{S})}$$

to these pairs of states using maximum likelihood estimation. Since the joint distribution $p_{\mathbf{S},\mathbf{S}_f}$ is Gaussian, the conditional distribution $p_{\mathbf{S}|\mathbf{S}_f}$ is also Gaussian with parameters $(\mu, \Sigma) = h(\mathbf{s}_f)$, where $h$ is the standard conditional Gaussian formula:

$$h(\mathbf{s}_f) = \left(\underbrace{\mu_{\mathbf{s}} + \Sigma_{\mathbf{ss}_f}\Sigma_{\mathbf{s}_f\mathbf{s}_f}^{-1}(\mathbf{s}_f - \mu_{\mathbf{s}_f})}_{\mu}, \underbrace{\Sigma_{\mathbf{ss}} - \Sigma_{\mathbf{ss}_f}\Sigma_{\mathbf{s}_f\mathbf{s}_f}^{-1}\Sigma_{\mathbf{s}_f\mathbf{s}}}_{\Sigma}\right). \tag{7}$$

In summary, given a final desired state $\mathbf{s}_f$, we generate a distribution by computing $\omega = h(\mathbf{s}_f)$ according to Equation 7. This conditional distribution also provides a simple way to relabel goal distributions given a reached state $\mathbf{s}_r$: we relabel the goal distribution by using the parameters $\omega' = h(\mathbf{s}_r)$.

**Multi-task exploration scheme** For training, in 50% of exploration rollouts we randomly selected a single subtask for the entire rollout, and in the other 50% of exploration rollouts we sequentially switched the subtask throughout the rollout, evenly allocating time to each subtask. For switching the subtask, we simply switched the parameters of the subtask $\mu$ and $\Sigma$. We randomized the order of the subtasks for sequential rollouts.

**Relabeling** We relabel the parameters of the goal distribution $(\mu, \Sigma)$, and the relabeling strategy we use depends on whether we use conditional goal distributions. For non-conditional distributions, we relabel according to the following strategy:

- 40%: relabel $\mu$ to a future state along the same collected trajectory

For conditional distributions, we relabel according to the following strategy:

- $40\%$: randomly sample $\mathbf{s}_f$ from the environment
- $40\%$: relabel $\mathbf{s}_f$ to a future state along the same collected trajectory

For our multi-task experiments, whenever we perform relabeling, we also relabel $\Sigma$. Specifically, we first randomly sample a task from the set of tasks that we have inferred, and relabel $\Sigma$ to the covariance matrix for that task.

### D.3 HER

Our implementation of goal-conditioned RL follows from hindsight experience replay (HER) [1]. Crucially, we perform off-policy RL, in addition to using the relabeling strategies inspired by HER. When provided a batch of data to train on, we relabel the goals according to the following strategy:

- $40\%$: randomly sampled goals from the environment, or the example sets
- $40\%$: future states along the same collected trajectory, as dictated by HER

Unlike HER, which used sparse rewards, we use the Euclidean distance as the basis for our reward function:

$$r(\mathbf{s}, \mathbf{s}_g) = -\left\|\mathbf{s} - \mathbf{s}_g\right\| \tag{8}$$

To avoid manual engineering, the space of goals is the same as the space of states. I.e., the dimension of the goal is the same as that of the state, and the corresponding entries in $\mathbf{s}$ and $\mathbf{s}_g$ correspond to the same semantic state features.

### D.4 VICE

Variational inverse control with events (VICE) is described in Fu et al. [15]. VICE proposes an inverse reinforcement learning method that extends adversarial inverse reinforcement learning (AIRL) Fu et al. [14]. Like AIRL, VICE learns a density $p_\theta(s, a)$ using a classification problem. However, unlike the usual IRL setting, VICE assumes access to an example set that specifies the task – the same assumption as DisCo RL.

VICE alternates between two phases: updating the reward and running RL. To learn a reward function, VICE solves a classification problem, considering the initial example set as positives and samples from the replay buffer as negatives. The discriminator is:

$$D_\theta(s, a) = \frac{p_\theta(s, a)}{p_\theta(s, a) + \pi(a|s)}. \tag{9}$$

At optimality, the reward recovered $p_\theta(s, a) \propto \pi^*(a|s) = \exp(A(s, a))$, the advantage of the optimal policy [14]. In practice the reward is represented as $p_\theta(s)$, ignoring the dependence on actions. However, actions are still needed to compute the discriminator logits; we follow the method specified in VICE-RAQ [45] to sample actions from $\pi(a|s)$ for all states. We also use mixup [52] as described in VICE-RAQ. Mixup significantly reduces overfitting and allows VICE to successfully learn a neural net classifier even with so few (30 to 50) positive examples. In the RL phase, VICE runs reinforcement learning with $\log p_\theta(s, a)$ as the reward function, actively collecting more samples to use as negatives.

We re-implemented VICE as above and confirmed that it successfully learns policies to reach a single state, specified by examples. However, our results demonstrate that VICE struggles to reach the example sets we use in this work. This issue is exacerbated when the problem is multi-task instead of single-task and the state includes a goal, as in goal-conditioned learning.

In multi-stage tasks, we train a single DisCo RL policy shared among the stages. For VICE, sharing data among different tasks would not respect the adversarial optimization performed by the method. Instead, we train separate policies for each stage without sharing data between policies. Thus, for a task with $N$ stages, VICE is generously allowed to experience $N\times$ the data, as each policy collects its own experience).