Notes from an ICML 2025 Tutorial: Generative AI Meets Reinforcement Learning

Benjamin Eysenbach
Assistant Professor of Computer Science
Princeton University
eysenbach@princeton.edu

Amy Zhang
Assistant Professor of Electrical and Computer Engineering
UT Austin
amy.zhang@austin.utexas.edu

Abstract

This tutorial explores the intersection of generative AI and reinforcement learning, demonstrating how generative models can be understood as RL agents and environments, and conversely, how RL can be viewed as generative modeling. It aims to bridge the gap between these fields, showing how insights from each can enhance the other. The tutorial will cover topics such as reinterpreting generative AI training through an RL lens, adapting generative AI to build new RL algorithms, and understanding how AI agents interacting with tools and humans create a new generative model. It will also discuss future directions and open problems, focusing on how RL can shape the future of foundation model training and enable generative AI systems to construct their own knowledge.

Recording and slides: https://generative-rl-tutorial.github.io/

Context: At ICML 2025, we (the authors) gave a tutorial on generative AI and reinforcement learning. We put these notes together as part of our preparation for the tutorial. These notes are not intended to be a comprehensive survey of generative AI or reinforcement learning. Rather, they are meant to highlight some ideas that we have found insightful and useful in our own research. These notes have not been subject to the scrutiny of peer review. Rather, they should be taken as a lightly-edited transcript of the tutorial itself.

1 Introduction

1.1 The Mechanical Turk, 1770

I want to start with a story about chess. Many of you are familiar with AlphaZero, DeepMind's chess-playing agent developed in 2018. Many may be familiar with IBM's DeepBlue, the chess-playing agent developed in 1995, which famously beat Gary Kasparov, a chess grandmaster. These were not the first attempts at developing automated systems for playing chess. Not even close.

I want to tell you about a chess-playing agent developed in the 70s (Fig. 1). The 1770s. Developed in Europe (Austria/Hungary, which were then the same nation?) by Wolfgang von Kempelen. Performed well against a wide array of human opponents, winning most of its games, including Napoleon and Benjamin Franklin.

Here's a picture of the system, along with the internal cogs that worked it.

How this story usually ends is by noting that those cogs are a façade. Literally. The way the machine worked is that a human was inside the box, pulling the levers to move the pieces. There was an elaborate set of mirrors so that you could open up the box and wouldn't see the guy inside.

People are disappointed once the secret is revealed because it means that there's no magic. The machine is no better than the human inside.

Figure 1: The cogs inside the Mechanical Turk had no real effect, and moves were instead decided by a human hidden behind mirrors inside the box. Reinforcement learning provides tools for building systems that can outperform even the best humans.

Zooming forward almost 250 years to today, there's a strong analogy to the generative models we have today: they are a fantastic feat of engineering, and impress many into thinking that they are truly intelligent. But once you peel back the mirrors, you realize that they are typically no smarter than the humans on which they are trained. Today, we have the technology to memorize the human's moves and encode them in cogs (in tiny silicon cogs, rather than copper ones). But most AI machines today are simply mimicking the humans they have been trained to mimic.

With imitation-based methods, we're only going to get as good as the best humans.

1.2 Roadmap for today

Today's tutorial is about reinforcement learning, a toolbox for building generative models that can do more than mimicry, a set of tools that allow you to build generative models that can solve tasks no human knows how to solve, not even experts. The tutorial will have four parts:

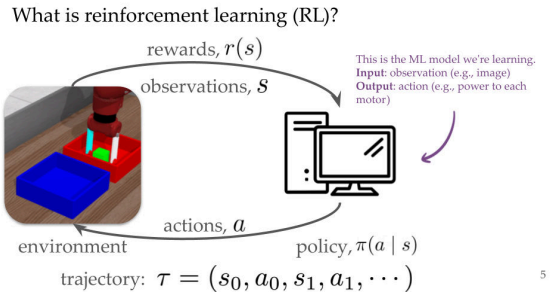
- Part I: using generative models as policies and world models
- Part II: interaction as a generative model, connections with goal reaching
- Part III: likelihoods of this generative model, fully-general RL problems
- Part IV: where does data come from
- Conclusion: what comes next

2 Part I: Using Generative Models in RL Engines

Today's tutorial is a story in 4 Parts. The first is going to be about how we can use existing generative models to build better reinforcement learning algorithms. This is the section that you've most likely thought about before, and it'll serve as a way to introduce the key concepts in RL.

2.1 What is reinforcement learning

Notation. We'll start by defining the reinforcement learning problem. We'll use states s_t, indexed by time $t = 0, \cdots$. We will consider both infinite and finite horizon settings. We'll use a_t to denote the action at time t. We'll use $r(s_t, a_t)$ to denote the reward at time step t; we'll also consider settings where the rewards just depend on the state, $r(s_t)$. We'll use $\tau = (s_0, a_0, \cdots)$ to denote a *trajectory* of experience, and define $R(\tau) \triangleq \sum_t \gamma^t r(s_t, a_t)$ as the discounted sum of rewards along this trajectory. The next state/observation will be sampled from the dynamics, $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$. Note that this can be both deterministic and stochastic. Finally, the initial state will be sampled $s_0 \sim p_0(s_0)$.



The goal in RL is to learn a policy $\pi(a_t \mid s_t)$ that takes as input the observation and outputs the action. The policy is trying to visit states and actions with high rewards. For example, in a game, these rewards correspond to points (e.g., in Tetris). For an assistive robot, rewards might be based on how well the robot enables the human to complete their tasks.

Supervised learning vs Reinforcement learning So, like in supervised learning, the policy is a mapping from inputs to outputs. Let's think about how this policy is trained. In supervised learning, we are given labeled examples, telling us what the correct label is for a given input. The key difference between supervised learning and reinforcement learning is that we're not given the correct label; we're not told what the correct action is in each setting.

As an example, consider the game of (say) tetris. One thing you could do is record the board position and moves of a bunch of players. This would give you a dataset of (s, a) pairs. You could then do supervised learning to mimic those data. *If you did this, how good would your policy be at beating humans?*

Now, if you had better human data, you'd get a better mimicry policy. But, at the end of the day, your policy would only be as good as the players that it was trained on. As the old adage goes, "garbage in, garbage out." This hopefully illustrate that mimicry is different from getting the best possible policy.

The key idea in reinforcement learning is to rethink the objective that we use. Instead of using mimicry as an objective, trying to find the policy that best imitates the actions that have been observed so far, we're going to use a different objective.

What is an objective? An objective is just a function that takes a policy and tells you how good that policy is. Before, when talking about mimicry, the objective was some notion of how well our policy replicated the actions from the data. A common way of writing this down in math is with the maximum likelihood objective:

$$\max_\theta \mathcal{J}(\pi) \qquad \text{where} \quad \mathcal{J}(\pi) \triangleq \mathbb{E}_{s,a\sim\mathcal{D}}[\log \pi_\theta(a \mid s)] \qquad \text{(supervised learning)}$$

and $\mathcal{D} = \{(s, a)\}$ is a dataset of states and corresponding actions taken by a human.

Let's walk through an example: a policy takes as input an observation s, and outputs an action distribution with mean $\mu_\theta(s)$ and variance 1: $\pi_\theta(a \mid s) = \mathcal{N}(a; \mu_\theta(s), \sigma = 1)$. In other words, the

action is computed as

$$a = \mu_\theta(s) + \epsilon, \qquad \text{where} \quad \epsilon \sim \mathcal{N}(0, 1).$$

The maximum likelihood loss then takes the following form:

$$\max_\theta \mathbb{E}_{s,a\sim\mathcal{D}}[-\frac{1}{2}\|a - \mu_\theta(s)\|_2^2 - \frac{d}{2}\log(2\pi)],$$

where $d = \dim(a)$. Note that this looks just like the MSE loss. Thus, we can understand mimicry as just trying to replicate the human's actions as closely as possible.

The RL Objective Reinforcement learning uses a different objective function $\mathcal{J}$. In particular, to evaluate how good/bad a policy is, reinforcement learning will measure how much return a given policy gets:

$$\mathcal{J}(\pi) \triangleq \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_t \gamma^t r(s_t, a_t)\right]. \qquad \text{(reinforcement learning)}$$

This objective is different from the supervised learning objective in a few key ways:

- The rewards are part of the problem definition, telling us how "good" each state and action are. The policy is trying to visit states and actions with high rewards. For example, in a game, these rewards correspond to points (e.g., in tetris). For an assistive robot, rewards might be based on how well the robot enables the human to complete their tasks.
- We sum these rewards across time. That is, we want a policy that not only gets us high returns right now, but also gets high returns in the future. Note that there can be a tradeoff here, in the same way that there's a tradeoff in eating vegetables today to make you happier tomorrow. The goal of the policy is not just to maximize rewards today, but also to put you in a spot so that tomorrow you can continue to get high rewards.
- There's a discount factor here, typically 0.9 or 0.99, some number just less than one. This discount means that, all else being equal, we'd prefer to get rewards sooner. If we'll only get one chocolate bar in our lives, we'd rather have it today than in a year. But, this is not to discount the previous point: we'd still prefer to get 2 chocolate bars tomorrow over getting 1 chocolate bar today.
- This objective doesn't depend on a dataset $\mathcal{D}$. The expectation in the RL objective is taken over trajectories sampled from π, not over data sampled from some human. It's worth sitting on the significance of this for a moment. In standard machine learning courses, the "goodness" of a model is based on how well it can predict labels for inputs, or produce samples similar to those in some given training data. But in RL, there is not data used in defining the problem. We'll come back to this point later.

This objective is quite powerful, as it lets you *specify* tasks that you don't know how to solve. E.g., beat a grandmaster at chess; fly an airplane to do a backflip; control a nuclear fusion reactor so that it produces more energy than it consumes.

2.2 Generative models as a world model

One of the key difficulties in optimizing this RL objective is that it requires interacting with an *environment*. The policy doesn't just spit out a number saying how good or bad it is. Rather, we have to plop the policy down in a particular environment (e.g., have it play a game, have it help a human), and only then can we look at the rewards and assess whether the policy is good or rubbish.

In some settings, this environment is pretty easy to use. In games, we can just have the policy play the game a bunch of times and see how it does. But in many potential applications of RL, from robotics to fusion to protein design, this environment isn't so easy to use. As one short example, we were working on a nuclear fusion project a year ago, and wrote a grant to use the facility, which was accepted: we were going to be allowed to use the facility for a few seconds.

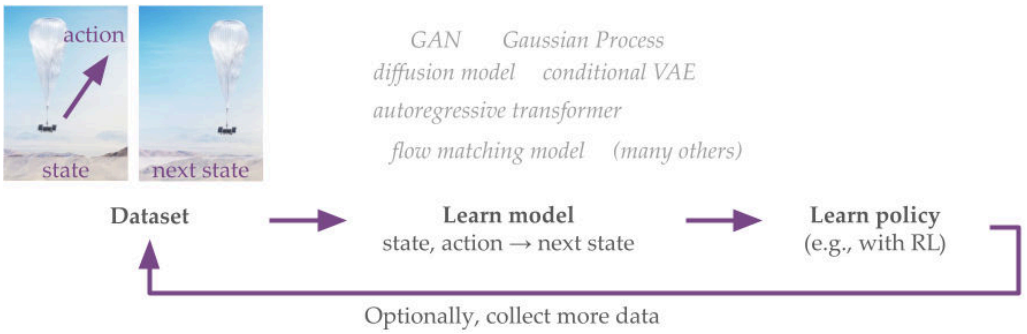


Figure 3: Overview of model-based RL.

What can we do? How can we learn good policies for important applications like this, when actually using the environment is really hard? Well, again, generative models may provide a key ingredient. Instead of using the real environment to figure out what happens when the policy takes a certain action, we can use a generative model to *simulate* what would happen. In jargon, we can use generative models as a "world model."

Figure 2: Two potential RL applications where simulation is difficult: education and nuclear fusion

Let's see how this works (see Fig. 3. This world-model generative-model is going to take as input a state and action and output a next state: $p_\theta(s' \mid s, a)$. We want that this model closely approximates the true environment dynamics:

$$p_\theta(s' \mid s, a) = p(s' \mid s, a) \qquad \text{for all} \quad s, a, s'.$$

How should we train this world model? Let's say we start with a bunch of data $\{(s, a, s')\}$. We can train the world model to mimic these data. Note that the world model isn't being trained with RL, but rather with maximum likelihood, so that it can be used as a stand-in for the real environment:

$$\max_\theta \mathbb{E}_{s,a,s' \sim \mathcal{D}}[\log p_\theta(s' \mid s, a)].$$

There are lots of generative models that one can choose from:

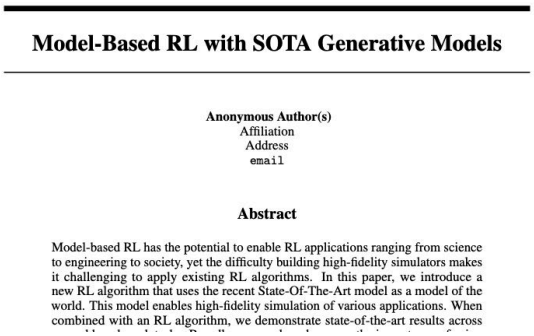
- Gaussian processes (Deisenroth and Rasmussen, 2011)
- Ensembles of Gaussian MLPs (Chua et al., 2018)
- (latent) State-space models (Hafner et al., 2019; Zhang et al., 2019)
- Diffusion models (Janner et al., 2022)
- Transformers (Chen et al., 2021; Janner et al., 2021)
- Flow matching models (Farebrother et al., 2025)
- ... and many others.

Challenges. There are several challenges associated with using a learned generative model as a proxy:

- Compounding errors (Ross et al., 2011; Venkatraman et al., 2015).
- Avoiding states where the model is uncertain (while still allowing the policy to explore) (Kidambi et al., 2020).
- Computational expense of running the model. Model-based RL algorithms are typically slow in terms of wall clock time.
- How and when to collect new data from the real environment (Janner et al., 2019).

And there are numerous papers tackling these challenges (Eysenbach et al., 2022a; Farahmand et al., 2017; Grimm et al., 2020; Lambert et al., 2020).

But there are also some key advantages of using a learned model. One nice thing about world models is that they run on GPUs, just like any other machine learning model. This means that you can run *batches* of simulations in parallel. This means that you can `jit` compile your data collection. So learning a world model is often appealing from an entirely computational perspective, and the model's errors are a reasonable cost to pay. Sometimes you might perform model-based RL even when you already have a simulator, simply because that simulator is too slow (e.g., for some games, weather simulation, nuclear fusion simulation).


Model-Based RL with SOTA Generative Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

Model-based RL has the potential to enable RL applications ranging from science to engineering to society, yet the difficulty building high-fidelity simulators makes it challenging to apply existing RL algorithms. In this paper, we introduce a new RL algorithm that uses the recent State-Of-The-Art model as a model of the world. This model enables high-fidelity simulation of various applications. When combined with an RL algorithm, we demonstrate state-of-the-art results across

Figure 4: One simple recipe for designing new RL algorithms is to use the newest generative models for model-based RL.

So, you want to write an ICML 2026 paper about RL? Here's a rough roadmap:

- Choose an existing RL algorithm (e.g., PPO, DQN)
- Choose a type of generative model for your policy
- Choose a type of generative model for your world model
- Plug-n-play
- Publish!

2.3 Where does data come from?

One of the most useful and interesting features of RL, as compared with supervised learning, is that it gives us mathematical tools for thinking about the provenance of data (see Fig. 5). In particular, it lets us think about data as part of the learning process. In supervised learning, the thing that you learn is the parameters of your NN. In RL, the thing that you learn is the parameters of your policy NN. But you're also learning something else – you're also exploring, gathering information about the world. We can think about exploration also as a learning process. In particular, we can think about your buffer of data as something that's getting optimized.

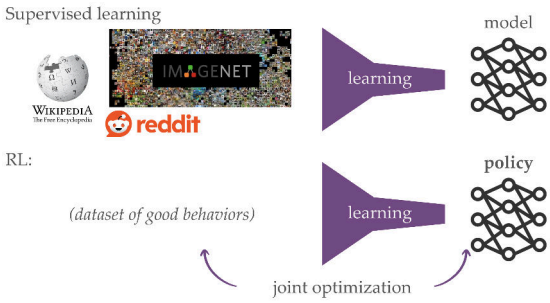


Figure 5: RL can be viewed as jointly optimizing the policy and the experience. This is different from supervised learning,w where data is provided as part of the input.

Let's see how this works. For simplicity, I'm going to write this out in terms of distributions over trajectories, τ, which are sequences of states and actions. We'll use $\pi_\theta(\tau)$ to denote the likelihood that policy π_θ visits the trajectory τ. Then, to think about the buffer/data, we'll use $q(\tau)$. Ideally, you'd have data from the optimal policy. If you had that data, then you could just mimic that data, and you'd be done. Conversely, if you had the optimal policy, then collecting this "optimal data" would be trivial – you'd just collect the data from your optimal policy. So, we can see that we have a sort of chicken-and-egg problem here. However, the generative AI perspective gives us important tools for thinking about how to resolve this.

Figure 6: RL can be viewed as a latent variable problem – inferring the unknown paths to high-reward states.

Let's think about RL from a probabilistic perspective – we want a policy that gets high returns, but we don't know which trajectories will gather those high returns (See Fig. 6). Thus, the trajectories

themselves are a *latent variable*. This is sort of similar to a VAE, where we're likewise missing part of the data – we don't know which z generated each example x, similar to how in RL we don't know which trajectories τ will generate high rewards. In the VAE, we have a trick for figuring out how to learn these latent variables – with an evidence lower bound. It turns out that we can do the same thing in RL (Peters et al., 2010; Peters and Schaal, 2007). We'll start by writing the log of the standard expected return objective; we'll also assume that the returns are positive:

$$\begin{aligned}\log \mathbb{E}_{\pi_\theta(\tau)}\left[R(\tau)\right] &= \log \int \pi_\theta(\tau) R(\tau) \frac{q(\tau)}{q(\tau} d\tau \\ &\geq \mathbb{E}_{q(\tau)}\left[\log \pi_\theta(\tau) + \log R(\tau) - \log q(\tau)\right].\end{aligned}$$

Let's unpack this. We started with an optimization problem over just the policy, π_θ. Then we obtained a lower bound on this objective that is a *joint optimization* over both the policy π_θ and the data $q(\tau)$. Recall that the ELBO holds for any choice of $q(\tau)$, and that we can optimize the ELBO w.r.t. $q(\tau)$. This means that we can jointly optimize this lower bound w.r.t. these two components.

Data optimization. Let's think about the optimization w.r.t. each of the components. Applying calculus of variations, we get that the optimal choice of $q(\tau)$ is a weighted version of $\pi_\theta(\tau)$:

$$q^*(\tau) = \frac{R(\tau)\pi_\theta(\tau)}{\int R(\tau')\pi_\theta(\tau')d\tau'}.$$

We can then plug this choice of $q(\tau)$ into the policy objective:

$$\begin{aligned}\max_{\pi_\theta} \mathbb{E}_{q^*(\tau)}\left[\log \pi_\theta(\tau) + \log R(\tau) - \log q^*(\tau)\right] &= \mathbb{E}_{q^*(\tau)}\left[\log \pi_\theta(\tau) + \cancel{\log R(\tau)} - \cancel{\log q(\tau)}\right] \\ &= \mathbb{E}_{\pi_\theta^{\text{old}}(\tau)}\left[R(\tau) \log \pi_\theta(\tau)\right].\end{aligned}$$

In the first line, I'm ignoring terms that don't depend on the *current* value of θ. Note that the expectation here is w.r.t. the policy at the last iteration.

Policy optimization. Now, to optimize this objective w.r.t. π_θ, we can take the gradient:

$$\begin{aligned}\nabla_\theta &= \mathbb{E}_{\pi_\theta^{\text{old}}(\tau)}[R(\tau)\nabla_\theta \log \pi_\theta(\tau)] \\ &= \mathbb{E}_{\pi_\theta^{\text{old}}(\tau)}\left[R(\tau)\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]. \qquad (1)\end{aligned}$$

So, the complete algorithm looks as follows:

- Sample trajectories from your policy
- Weight those trajectories by their return
- To maximum likelihood on the reward-weighted trajectories.

This method is also known as *reward-weighted regression* (Peters and Schaal, 2007), because you're "regressing" your policy onto the reward-weighted trajectories.

The gradient of this last expression in Eq. 1 is closely related to the *policy gradient* (Williams, 1992), and its one of the most central objects in reinforcement learning. Usually, it is derived from a very different perspective, from calculus or stochastic optimization. But we see that the probabilistic perspective on RL provides an alternative and complementary way of getting this same result.

There are lots and lots of more sophisticated versions of this. But one thing I'd like to emphasize is that this is not a heuristic. We're not designing some custom dataset curation scheme, we're not doing some ad-hoc thing to filter data or reweight data. Rather, elementary algebra tells us exactly what sorts of updates/gradients to compute to find policies that select actions that maximize reward.

Summary. So, to answer the question posed at the start of this subsection, data comes from the policy itself, after reweighting. Now that we know where data is coming from, we can use it to train the policy. What data should you use to learn a model? Well, you can think of $q(\tau)$ as a

model, and derive new model-based algorithms that are optimized not for accuracy, but rather to produce policies that get high returns (Eysenbach et al., 2022a; Ghugare et al., 2023)!

Now that we have some clues on how to train models, we'll move on to thinking about how generative models might be used as policies.

2.4 Generative models as Policies

There are various ways in which existing generative models can be plugged into RL algorithms as the policy. We will divide these into two forms: reward-conditioned reinforcement learning vs. the more standard policy gradient and actor-critic approaches.

In reward-conditioned reinforcement learning, transformers and diffusion models (Chen et al., 2021; Janner et al., 2022, 2021) have been used with an autoregressive objective to train on sequential data consisting of the states, actions, and rewards:

$$S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, \ldots$$

You can think of this as training a generative model to simultaneously be a world model (next section) and a policy. Let's focus on the policy version. We can incorporate masking to train the generative model to only output actions:

$$S_t, \_\_, R_{t+1}, S_{t+1}, \_\_, \ldots$$

but this only allows us to train an agent that can reproduce the same behavior as what is in the dataset. How can we do better? By using context conditioning in an interesting way. Instead of just learning a policy $\pi(a|s)$, the model learns a conditioned policy $\pi(a|s, r)$, where r is the target return (desired reward).

One of the reasons why I wanted to go through the basics, and do it in this way, is to highlight that $\pi_\theta(a \mid s)$ is just a particular choice of a generative model. While we used a Gaussian policy as an example, none of the subsequent math was predicated on a particular type of policy. Rather, we can plug in any policy we'd like.

Let's go back to this gradient again:

$$\nabla_\theta \mathcal{J}(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[R(\tau) \sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right].$$

The term $\nabla_\theta \log \pi_\theta(a \mid s)$ is known as the *score function*, the same score function that is ubiquitous in generative modeling. So, for any generative model, as long as we can easily compute its score function, we can plug it into this update to create an RL method.

But, as we'll see in the following, there's some nuance in doing this effectively. Below, we'll walk through a concrete case study of what it takes to take a performant generative model and adapt it to be a reward-maximizing policy.

2.5 A Case Study: Diffusion-DICE

Diffusion models (Ho et al., 2020; Sohl-Dickstein et al., 2015; Song et al., 2020) are generative models based on a Markovian noising and denoising process. Given a random variable x_0 and its corresponding probability distribution $q_0(x_0)$, the diffusion model defines a forward process that gradually adds Gaussian noise to the samples from x_0 to $x_T (T > 0)$. Kingma et al. (2021) shows there exists a stochastic process that has the same transition distribution $q_{t0}(x_t|x_0)$ and Song et al. (2020) shows that under some conditions, this process has an equivalent reverse process from T to 0. The forward process and the equivalent reverse process can be characterized as follows, where $\bar{\mathbf{w}}_t$ is a standard Wiener process in the reverse time.

$$q_{t0}(x_t|x_0) = \mathcal{N}(x_t|\alpha_t x_0, \sigma_t^2 \mathbf{I}) \quad dx_t = [f(t)x_t - g^2(t)\nabla_{x_t} \log q_t(x_t)]dt + g(t)d\bar{\mathbf{w}}_t, \; x_T \sim q_T(x_T). \tag{2}$$

Here we slightly abuse the subscript t to denote the diffusion timestep. α_t, σ_t are the noise schedule and $f(t)$, $g(t)$ can be derived from α_t, σ_t (Lu et al., 2022). For simplicity, we denote $q_{t0}(x_t|x_0)$ and $p_{0t}(x_0|x_t)$ as $q(x_t|x_0)$ and $p(x_0|x_t)$, respectively. To sample from $q_0(x_0)$ by following the reverse stochastic differential equation (SDE), the score function $\nabla_{x_t} \log q_t(x_t)$ is required.

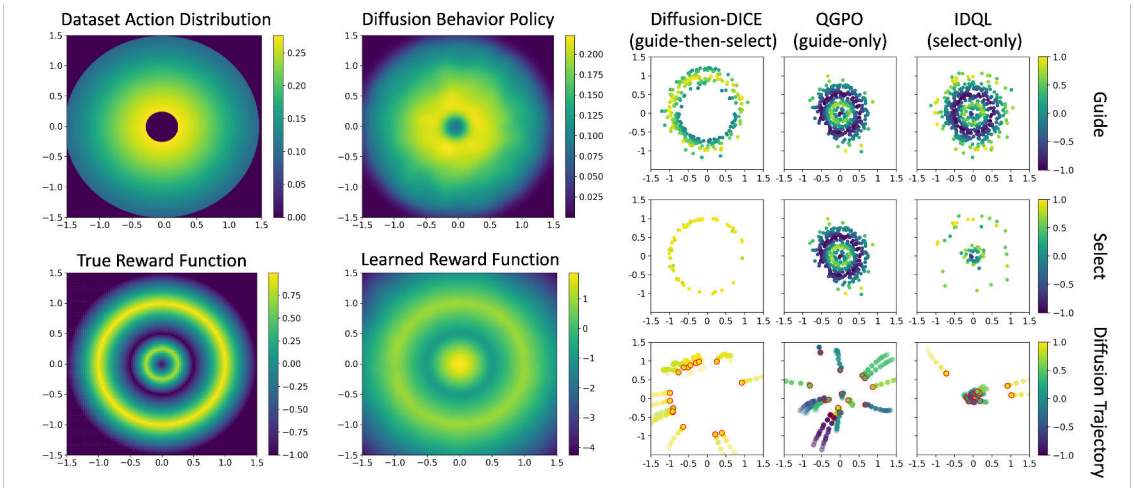


Figure 7: Toycase of a 2-D bandit problem. The action in the offline dataset follows a bivariate standard normal distribution constrained within an annular region. The ground truth reward has two peaks extending from the center outward. We use a diffusion model $\hat{\pi}^{\mathcal{D}}$ to fit the behavior policy and a reward model $\hat{R}$ to fit the ground truth reward R. Both $\hat{\pi}^{\mathcal{D}}$ and $\hat{R}$ fit in-distribution data well while making error in out-of-distribution regions. Diffusion-DICE could generate correct optimal actions in the outer circle while other methods tend to expolit error information from $\hat{R}$ and only generate overestimated, sub-optimal actions.

Typically, diffusion models use denoising score matching to train a neural network $\epsilon_\theta(x_t, t)$ that estimates the score function (Ho et al., 2020; Song et al., 2020; Vincent, 2011), by minimizing $\mathbb{E}_{t\sim\mathcal{U}(0,T),x_0\sim q_0(x_0),\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}[\|\epsilon_\theta(x_t,t)-\epsilon\|^2]$, where $x_t = \alpha_t x_0 + \sigma_t\epsilon$. As we mainly focus on diffusion policy in RL, this objective is usually impractical because $q_0(x_0)$ is expected to be the optimal policy $\pi^*(a|s)$.

To make diffusion models compatible with RL, there are generally two approaches: *guide-based* and *select-based*. Guide-based methods (Janner et al., 2022; Lu et al., 2023) incorporate the behavior policy's score function with an additional guidance term. Specifically, they learn a time-dependent guidance term $\mathcal{J}_t$ and use it to drift the generated actions towards high-value regions. The learning objective of $\mathcal{J}_t$ can be generally formalized with $\mathcal{L}(\mathcal{J}_t(a_t), w(s, \{a_0^i\}_{i=1}^K)))$, where $w(s, \{a_0^i\}_{i=1}^K)$ are critic-computed values on K diffusion behavior samples. $\mathcal{L}$ can be a contrastive objective (Lu et al., 2023) or mean-square-error objective (Janner et al., 2022). After training, the augmented score function $\nabla_{a_t}\log\pi_t(a_t|s) = \nabla_{a_t}\log\pi_t^{\mathcal{D}}(a_t|s) + \nabla_{a_t}\mathcal{J}_t(a_t)$ is used to characterize the learned policy.

Select-based methods (Chen et al., 2022; Hansen-Estruch et al., 2023) utilize the observation that for some RL algorithms, the actor induced through critic learning manifests as a reweighted behavior policy. To sample from the optimized policy, these methods first sample multiple candidates $\{a_0^i\}_{i=1}^N$ from the diffusion behavior policy and then resample from them using critic-computed values $w(s, \{a_0^i\}_{i=1}^N)$. More precisely, the sampling procedure follows the categorical distribution $Pr[a = a_0^j|s] = \frac{w(s,a_0^j)}{\sum_{i=1}^N w(s,a_0^i)}$.

Error exploitation. As we can see, both guide-based and select-based methods need the information of $w(s, \{a_0^i\}_{i=1}^N)$ to get the optimal action. However, this term may bring two sources of errors. One is the diffusion model's approximation error in modeling complicated policy distribution, and the other is the critic's error in evaluating unseen actions. Although trained on offline data, the diffusion model may still generate OOD actions (especially with frequent sampling) and the learned critic can make erroneous predictions on these OOD actions, causing the evaluated value on these actions to be over-estimated due to the learning nature of value functions (Fujimoto et al., 2019; Kumar et al., 2019). As a result, the generation of high-quality actions in existing methods is

greatly affected due to this error exploitation, which we will also show empirically in the next section.

Guide-Then-Select Paradigm. We developed Diffusion-DICE (Mao et al., 2024a), which uses both guide and select steps to more accurately select high-value actions. We first train a diffusion model to fit the behavior policy's score function. Next, it turns out that the optimal policy's score = behavior policy's score + the gradient of a guidance term derived from the optimal distribution ratio.

For the guide step, we use both score components to sample candidate actions. Due to the multi-modality contained in the optimal policy distribution, the guide-step may guide towards those local-optimal modes. We thus generate a few candidate actions and use the Q function to select one with the highest value to achieve the global optimum.

We use a toycase to validate that the *guide-then-select* paradigm used in Diffusion-DICE (Mao et al., 2024a) indeed brings minimal error exploitation that other diffusion-based methods suffer from. Here we aim to solve a 2-D bandit problem given a fixed action dataset. The action space is continuous and actions in the offline dataset follow a bivariate standard normal distribution constrained within an annular region. We show the dataset distribution $\pi^{\mathcal{D}}$, the learned diffusion behavior policy $\hat{\pi}^{\mathcal{D}}$, the ground truth reward R and the predicted reward $\hat{R}$ in Figure 7. Note that the true optimal reward occurs on the outer circle. However, due to limited data coverage, the learned reward function exploits overestimation error on unseen regions, e.g., actions inside the inner circle have erroneous high values. What's worse, due to fitting errors, the diffusion behavior policy may generate such overestimated actions. We take Diffusion-DICE with a guide-based method, QGPO (Lu et al., 2023), and a select-based method, IDQL (Hansen-Estruch et al., 2023) for comparison.

2.6 Diffusion Models from an RL Perspective

Conversely, we can train diffusion models with reinforcement learning objectives (Black et al., 2024b). Denoising Diffusion Policy Optimization (DDPO), is a novel way to optimize diffusion models for downstream objectives. They reframe the diffusion denoising process as a multi-step Markov decision process (MDP). Each denoising step becomes an RL action, and the reward is given at the end, based on sample quality (e.g., aesthetics, compressibility, prompt alignment). They show that you can use policy gradient estimators (a few types) and outperform a simpler method of reward-weighted regression (RWR), which approximates reward-weighted likelihoods but suffers from poor optimization and approximation issues.

Segue to next chapter. Zooming out a whole lot, what I hope that this has highlighted is that, in addition to plugging generative models into RL algorithms in rather straightforward ways, this generative modeling perspective can also reveal some pretty surprising new ways of using and training generative models. The key idea there was to think about the RL problem as a whole really as a sort of generative modeling problem.

In this next part, we're going to take this one step further: in the same way that generative models today are trained with *data* as supervision, how can we train RL algorithms using data (not rewards) as supervision?

3 Part II: Interaction as a Generative Model

In this section, we'll introduce a new sort of generative model that ties RL and generative modeling even more closely together. In the previous section, we talked about how your policy can be a generative model (of actions) and how your world model can be a generative model (or observations). In this section we'll show how your policy interacting with the environment forms a new generative model. In the same way that you can sample from (say) a video generative model, you'll be able to sample from this agentic generative model. And in the same way that you can compute likelihoods of generative models, we'll show that you can estimate the likelihoods of outcomes under this model – these likelihoods will correspond to the value function.

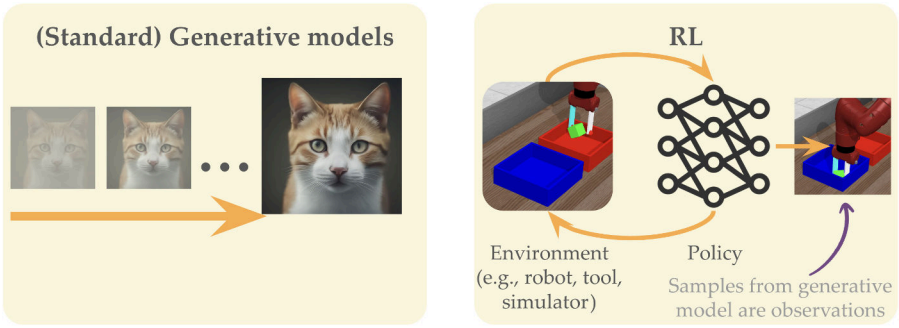


Figure 8: **RL as generative AI.** *(Left)* Generative models perform iterative computation (e.g., denoising steps) to eventually arrive at a good sample. *(Right)* RL methods are doing something similar: doing interaction to eventually arrive at a good observation. RL can be seen as a generative model, which interleaves policy steps with environment steps. In Sec. 3, we think about a specific type of RL problem, goal-reaching problems, where tasks are specified by data, mirroring how tasks are specified for other generative models.

In this section, we'll then concretely look at goal-reaching as a case study of how such a perspective is useful. But then, after the break, we'll talk about how this perspective is fully general.

3.1 Your RL Agent is a Generative Model in Disguise.

Let's go back to basics: what is a generative model? We usually think about a generative model as something that can (1) generate samples from some distribution, and/or (2) estimate the likelihood of samples. Examples might be a language model, a diffusion model, a VAE, even a Gaussian mixture model.

For several decades, the best generative models are those that perform *iterative computation* to produce samples. While a GMM samples a cluster and then adds noise to the cluster mean, more powerful and capable generative models all perform iterative computation to produce samples. For example, VAEs and normalizing flows and GANs take as input a noise vector and apply a long sequence of NN layers to transform that noise vector into (say) a natural image. Diffusion models and flow-matching models take a noise vector and iteratively subtract noise from that image.

We usually think about this iterative computation as happening with known, differentiable functions (think: layers of a neural network). But that need not be the case. Take, for example, the problem of rendering: given a configuration of objects and lights in a scene, generate an image of that scene. Such a render still seems like a generative model, even though it may not be differentiable, and even though it may not be implemented with PyTorch primitives. Indeed, there's a long line of work in cosmology and weather prediction (among other fields) treating black-box models as a generative model; you can still do inference with these methods using techniques like approximate Bayesian computation (Csilléry et al., 2010), simulation-based inference (Cranmer et al., 2020), and neural posterior inference (Dax et al., 2021). The fact that the black boxes are or are-not differentiable seems beyond the point.

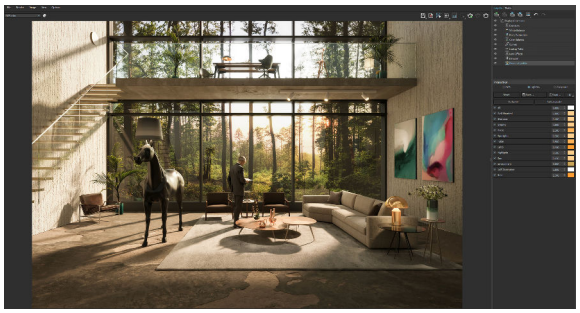


Figure 9: If we can think of rendering (e.g., Blender) as a sort of generative model, then maybe we can also think about RL as a sort of generative model.

We're arguing to take this one step further: treating an RL agent interacting with a (black-box) environment as a generative model. Some parts of the generative process are differentiable – we know the likelihood of sampling an action given an observation. Some parts of the generative process may not be differentiable – we don't necessarily know the environment dynamics.

So, here's the key idea: we'll treat RL agents interacting with an environment as a generative process. Samples from this model are observations at some point in time. This generative model can be made conditional by prompting the policy with some extra information. Like other generative models, the "samples" from the generative model are now the product of a long sequence of computation, some of it performed by learned layers, and some of it performed by other agents or tools.

OK, so why is this useful and what can you do with it?

What can you do? Starting in Sec. 3.2, we'll talk about how this perspective on RL is particularly natural for goal-reaching tasks, where the aim is to reach (sample!) a certain goal state at some point in the future. Then, Sec. 4 will extend the discussion to fully-general reward functions.

Why is this useful? We'll offer three reasons.

1. **A principled way to think about agents interacting with humans, tools, and their environment.** Perhaps the standard way of thinking about LLMs is that they are trained to do reasoning and thinking. These capabilities are entirely "imaginary" in the sense that they purely happen in the "mind" of an LLM – they do not actually change the world. I think the reason a lot of us are in this room is because we believe that AI can make the world better. Some of this can be done by using AI to better understand the world. But one hope is that we might use AI to directly change the world for the better, whether via assistants (interacting with humans) or the physical world (e.g., routing trains, recommending medical treatments). Generative models seem the most promising route to realizing these capabilities, yet they lack a formal language for describing interaction. RL provides this formal language. Treating interaction as a generative process tells us how to leverage generative models using the language and structures of RL.
2. **Reward engineering is hard.** There have been many amazing success stories in the RL community, and there have been lots of great examples of using RL to build better vision and language models. However, these papers typically reduce the problem to one of scalar reward maximization. And, that reward function typically has to be defined by a human. Reward engineering is really hard.

   The standard foundation of RL, whether from a stochastic optimization perspective or an optimal control perspective, is built upon rewards. Yet, we often don't think about where these rewards come from. In the standardized benchmarks, they are just there, so we typically don't think about why they are the way they are, or whether they actually make sense.

   Lots of reward engineering goes into many tasks (see, e.g., Fig. 10). There are lots of hyperparameters. We see that the reward function is doled out in stages; these stages effectively reflect how a human expects the task to be solved. But the whole point of RL is that it enables us to solve tasks that we don't know how to solve, or that it will find solutions better than the currently-conceptualized solutions. So, designing reward functions like this is problematic.

   While this discussion often becomes one of "sparse vs dense" rewards, there are gradations of sparsity. Often, you'll see a reward function that gives a "sparse" +1 upon completing each stage of the task. But the fact that the human designer split the task up into stages still constitutes a non-trivial degree of reward shaping.

   In this section, we'll think about a different way of framing some RL problems, which will also resolve a certain incongruence with the standard MDP definition.
3. There's one bit about the "RL as GenAI" story that still doesn't quite fit right: the reward function. When we think about generative models, we think about them being trained on *data*. But when we think about RL (e.g., the RL objective above), we are defining the problem in terms of this scalar reward. When we want to teach an image classifier to identify cats and dogs, we give it data, a bunch of images accompanied by labels (more

data). Yet when we want to teach a policy how to play Pong, we have to write a bunch of code that assigns scalar values to each observation? These reward functions make it challenging for non-experts (and even experts!) to use RL in practice. It also makes the entire RL enterprise somewhat dissonant with other areas of ML. In this part, we're going to resolve this dissonance by reframing the RL problem in a way that does away with reward functions.

3.2 Why Learn to Reach Goals?

OK, so the main aim of this section is to think about a certain special case of the RL problem: that of reaching a certain goal. A goal will be an observation. For example, if you're a mouse navigating a maze, instead of getting a reward of +1 when you get to a certain location, we'll think about telling the mouse that that location is its goal.

Reaching goals is a classic problem dating back to the early days of AI/ML (Kaelbling, 1993; Laird et al., 1987; Newell and Simon, 1961).

Thinking in terms of goals allows us to eschew reward functions. As we'll see in this section, we nonetheless get many coherent algorithms, including ones that can perform effective exploration and long-horizon reasoning.

Goal-reaching has many practical applications:

- Robot locomotion and manipulation (Team et al., 2023)
- Autonomous driving
- Chemical synthesis (Beeler et al., 2023; Zhou et al., 2017)
- Path planning, route optimization
- Games (e.g., build a castle in Minecraft (Baker et al., 2022; Milani et al., 2024)).

Intuitively, we want to get to the goal as quickly as possible. We'd also like to stay there as long as we can. A natural way to think about this is that you get a +1 when you're at the goal and +0 otherwise. This is well defined in settings where you have a finite number of states. However, this reductionist approach has a couple limitations. First, it seems to make the RL problem more difficult – you've just specified a hard-to-optimize reward function. How are you supposed to actually learn to maximize this reward function? Second, it doesn't seem to provide any insight into how to actually solve goal-reaching problems.

So, in the coming sections, we're going to think about goal-reaching in a different way: in terms of probabilities. Instead of thinking about the *time* you spend at the goal, we'll think about the *probability* that you get to the goal. Of course, in settings with a finite number of states, these are exactly the same. But thinking in terms of probabilities opens up new avenues of inquiry, new ways of analyzing this problem, new ways of solving it. In particular, the generative AI revolution was largely predicated on a probabilistic understanding of data. We'll see that taking this probabilistic perspective allows us to better harness the tools of generative AI to solve difficult problems; and, we'll even see how it allows us to do the opposite, using the tools of RL to train generative models in better ways.

Roadmap. We'll start by thinking about estimating the probability of reaching states at some point in the future. We'll see how these probabilities can be used to determine the actions for reaching desired goal states. This will give us an effective method for goal-conditioned RL.

3.3 Defining the goal-reaching problem

We will look at the following objective for goal-reaching:

$$\max_{\pi}(1-\gamma)\mathbb{E}_{\pi}\left[\sum_{t}\gamma^{t}\mathbb{1}(s_t=g)\right].$$

The $(1-\gamma)$ constant factor doesn't affect the optimization problem. We include it to make our analysis easier later on. In continuous settings, you never get to a state exactly (this is a measure

zero event), so instead we look at the probability that the *next* state is the goal:

$$\max_{\pi}(1-\gamma)\mathbb{E}_{\pi}\left[\sum_{t}\gamma^{t}p(s_{t+1}=g \mid s_t, a_t)\right]. \tag{3}$$

Note that this objective is well defined in both continuous and discrete settings, and in both stochastic and deterministic settings. In continuous settings, we will talk about probability *densities*.

Intuition. Intuitively, we can understand this objective as saying that you get a +1 when you're at the goal and zero otherwise. We can think about this as maximizing the total time you spent at the goal. Like the standard RL problem, we'd prefer to get to the goal quickly, which is why we include the discount factor of γ. This objective looks like the standard RL problem with a reward function

$$r_g(s,a) = (1-\gamma)p(s'=g \mid s,a). \tag{4}$$

The nice thing about this problem formulation is that the rewards are defined directly in terms of the data (see Fig. 10). Also, by defining this objective in terms of probabilities, it will have close connections with probabilistic modeling that we've seen in other areas of ML.

Challenges with goal-reaching. Eq. 4 shows how we've reduced this problem of probability matching or goal-reaching into a problem of RL. However, the RL problem itself is very hard. And, applying it to this reward requires estimating this reward function, which would require fitting a density model to the dynamics, which is challenging in high-dimensional settings. So, we're going to need a way of estimating and optimizing the objective in Eq. 3 that doesn't require estimating densities over high-dimensional objects. Perhaps surprisingly, this is doable, and not too hard! But first, let's start with some background on the dual formulation for RL.

```
def compute_reward(observation):
    objPos = obs[3:6]
    (rightFinger, leftFinger) = (self._get_site_pos('rightEndEffector'),
                                 self._get_site_pos('leftEndEffector'))
    fingerCOM = (rightFinger + leftFinger) / 2
    heightTarget = self.heightTarget
    placingGoal = self._target_pos
    reachDist = np.linalg.norm(objPos - fingerCOM)
    placingDist = np.linalg.norm(objPos[:2] - placingGoal[:-1])
    def reachReward():
        reachRew = -reachDist
        reachDistxy = np.linalg.norm(objPos[:-1] - fingerCOM[:-1])
        zRew = np.linalg.norm(fingerCOM[-1] - self.init_fingerCOM[-1])
        if reachDistxy < 0.06:
            reachRew = -reachDist
        else:
            reachRew = -reachDistxy - zRew
        if reachDist < 0.05:
            reachRew = -reachDist + max(actions[-1], 0) / 50
        return (reachRew, reachDist)
    def pickCompletionCriteria():
        tolerance = 0.01
        if objPos[2] >= heightTarget - tolerance:
            return True
        else:
            return False
    if pickCompletionCriteria():
        self.pickCompleted = True
    def objDropped():
        return objPos[2] < self.objHeight + 0.005 and placingDist \
            > 0.02 and reachDist > 0.02
    def placeCompletionCriteria():
        if abs(objPos[0] - placingGoal[0]) < 0.05 and abs(objPos[1]
            - placingGoal[1]) < 0.05 and objPos[2] < self.objHeight + 0.05:
            return True
        else:
            return False
// Yu et al. "Meta-world: A benchmark and evaluation for multi-task and meta
reinforcement learning." CoRL, 2020
```

Figure 10: Designing reward functions is challenging.

3.4 Introduction to Dual RL

For the first step, we consider a Markov decision process (MDP) *without* reward function defined by states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, initial state distribution $p_0 \in \Delta(\mathcal{S})$, discount factor $\gamma \in (0,1]$, and dynamics $p : \mathcal{S} \times \mathcal{A} \mapsto \Delta(\mathcal{S})$, where $\Delta(\cdot)$ denotes the probability simplex.

Discounted state occupancy measure. One mathematical objective that will be useful throughout today's tutorial is the *discounted state occupancy measure* (Puterman, 2014; Syed et al., 2008). Imagine that you close your eyes and open them after a random number of time steps. You want to know which state the agent will be at. Formally, this probability is referred to as the discounted state occupancy measure:

$$p^{\pi}(s_f) \triangleq (1-\gamma)\mathbb{E}_{\pi}\left[\sum_{t}\gamma^{t}p(s_{t+1}=g \mid s_t, a_t)\right]. \tag{5}$$

The $(1-\gamma)$ out front is to ensure that this whole thing sums/integrates to one. This is the probability of reaching state s_f at some point in the future. We will sometimes think about the conditional version of this, $p(s_f \mid s, a)$, which tells us the states we'll visit at some point starting at a particular state s and action a.

RL optimizes the expected return of a policy. We consider the linear programming formulation of the expected return (Manne, 1960), to which we can apply Lagrangian duality or Fenchel-Rockfeller duality to obtain corresponding constraint-free problems. We now review this framework, first introduced by Nachum and Dai (2020)[1]. Consider the regularized policy learning problem

$$\max_{\pi} J(\pi) = \mathbb{E}_{d^{\pi}(s,a)}[r(s,a)] - \alpha D_f(d^{\pi}(s,a) \mid\mid d^{O}(s,a)), \tag{6}$$

where $D_f(d^{\pi}(s,a) \mid\mid d^{O}(s,a))$ is a conservatism regularizer that encourages the visitation distribution of π to stay close to some distribution d^O, and α is a temperature parameter that balances the expected return and the conservatism. An interesting fact is that $J(\pi)$ can be rewritten as a convex problem that searches for a visitation distribution that satisfies the *Bellman-flow* constraints. We refer to this form as primal-Q:

$$\begin{aligned} &\text{primal-Q} \quad \max_{\pi} J(\pi) = \max_{\pi} \big[\max_{d} \mathbb{E}_{d(s,a)}[r(s,a)] - \alpha D_f(d(s,a) \mid\mid d^{O}(s,a)) \\ &\text{s.t } d(s,a) = (1-\gamma) d_0(s).\pi(a|s) + \gamma \textstyle\sum_{s',a'} d(s',a') p(s|s',a') \pi(a|s), \ \forall s \in \mathcal{S}, a \in \mathcal{A} \big]. \end{aligned} \tag{7}$$

We can convert this to an unconstrained problem with dual variables $Q(s,a)$ defined for all $s, a \in \mathcal{S} \times \mathcal{A}$ by applying Lagrangian duality and the convex conjugate, giving us the dual-Q formulation:

$$\text{dual-Q} \quad \max_{\pi} \min_{Q} (1-\gamma) \mathbb{E} s \sim d_0, a \sim \pi(s) Q(s,a) + \alpha \mathbb{E}(s,a) \sim d^{O} f^{*}\left(\left[\mathcal{T}_r^{\pi} Q(s,a) - Q(s,a) \right] / \alpha \right), \tag{8}$$

where f^* is the convex conjugate of f. In fact, one can note that Problem equation 7 is overconstrained—the constraints already determine the unique solution d^{π}, rendering the inner maximization w.r.t d unnecessary. Therefore, we can relax the constraints to obtain another problem with the same optimal solution π^* and d^*, which we call primal-V below:

$$\begin{aligned} &\text{primal-V} \quad \max_{d \geq 0} \mathbb{E}_{d(s,a)}[r(s,a)] - \alpha D_f(d(s,a) \mid\mid d^{O}(s,a)) \\ &\text{s.t } \textstyle\sum_{a \in \mathcal{A}} d(s,a) = (1-\gamma) d_0(s) + \gamma \sum_{(s',a') \in \mathcal{S} \times \mathcal{A}} d(s',a') p(s|s',a'), \ \forall s \in \mathcal{S}. \end{aligned} \tag{9}$$

Similarly, we consider the Lagrangian dual of equation 9, with dual variables $V(s)$ defined for all $s \in S$:

$$\text{dual-V} \quad \min_{V} (1-\gamma) \mathbb{E} s \sim d_0 V(s) + \alpha \mathbb{E}(s,a) \sim d^{O} f_p^{*}\left(\left[\mathcal{T} V(s,a) - V(s) \right] / \alpha \right), \tag{10}$$

where f_p^* is a variant of f^* defined as $f_p^*(x) = \max(0, {f'}^{-1}(x))(x) - f(\max(0, {f'}^{-1}(x)))$. This modification is to cope with the nonnegativity constraint $d(s,a) \geq 0$ in primal-V. Note that in both cases for dual-Q and dual-V, the optimal solution is the same as their primal formulations due to strong convexity.

Remarks. The dual formulations have a few appealing properties. (a) They allow us to transform constrained distribution-matching problems into unconstrained forms w.r.t previously logged data. (b) One can show that the gradient of dual-Q w.r.t π, when Q is optimized for the inner problem, is the on-policy policy gradient computed by off-policy data (Nachum and Dai, 2020). This property is key to relieving the instability or divergence issue in many off-policy learning algorithms (Fujimoto et al., 2018; Thrun and Schwartz, 1993).

What are interesting ways we can use generative models directly into more traditional RL methods? If we think about dual methods, there is a class of dual methods called DICE methods. DICE methods (Sikchi et al., 2023) incorporate the LP form of expected return $\mathcal{J}(\pi) = \mathbb{E}_{(s,a) \sim d^{\pi}}[r(s,a)]$ with a regularizer $D_f(d^{\pi} || d^{\mathcal{D}}) = \mathbb{E}_{(s,a) \sim d^{\mathcal{D}}}[f(\frac{d^{\pi}(s,a)}{d^{\mathcal{D}}(s,a)})]$, where D_f is the f-divergence induced by a convex function f (Boyd et al., 2004). More specifically, DICE methods try to find an optimal policy π^* that satisfies:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{(s,a) \sim d^{\pi}}[r(s,a)] - \alpha D_f(d^{\pi} || d^{\mathcal{D}}). \tag{11}$$

This objective is generally intractable due to the dependency on $d^{\pi}(s,a)$, especially under the offline setting. However, by imposing the *Bellman-flow* constraint (Manne, 1960) $\sum_{a \in \mathcal{A}} d(s,a) =$

[1]We use Lagrangian duality instead of Fenchel-Rockfeller duality for ease of exposition.

$(1-\gamma)d_0(s) + \gamma \sum_{(s',a')} d(s',a')p(s|s',a')$ on states and applying Lagrangian duality and convex conjugate, its dual problem has the following tractable form:

$$\min_V (1-\gamma)\mathbb{E}_{s\sim d_0}[V(s)] + \alpha \mathbb{E}_{(s,a)\sim d^{\mathcal{D}}}[f^*([\mathcal{T}V(s,a) - V(s)]/\alpha)]. \tag{12}$$

Here f^* is a variant of f's convex conjugate and $\mathcal{T}V(s,a) = r(s,a) + \gamma \mathbb{E}_{s'\sim p(\cdot|s,a)}[V(s')]$ represents the Bellman operator on V. In practice, one often uses a prevalent semi-gradient technique in RL that estimates $\mathcal{T}V(s,a)$ with $Q(s,a)$ and replaces the initial state distribution d_0 with dataset distribution $d^{\mathcal{D}}$ to stabilize learning (Mao et al., 2024b; Sikchi et al., 2023). In addition, because $\mathcal{D}$ usually cannot cover all possible s' for a specific (s,a), DICE methods only use a single sample of the next state s'. The update of $Q(s,a)$ and $V(s)$ in DICE methods are as follows:

$$\begin{aligned} &\min_V \ \mathbb{E}_{(s,a)\sim d^{\mathcal{D}}}\left[(1-\gamma)V(s) + \alpha f^*\left([Q(s,a) - V(s)]/\alpha\right)\right] \\ &\min_Q \ \mathbb{E}_{(s,a,s')\sim d^{\mathcal{D}}}\left[\left(r(s,a) + \gamma V(s') - Q(s,a)\right)^2\right]. \end{aligned} \tag{13}$$

Note that learning objectives of DICE-methods can be calculated solely with a (s,a,s') sample from $\mathcal{D}$, which is totally in-sample. One important property of DICE methods is that Q^* and V^* have a relationship with the optimal stationary distribution ratio $w^*(s,a)$ as

$$w^*(s,a) := \frac{d^*(s,a)}{d^{\mathcal{D}}(s,a)} = \max\left(0, (f')^{-1}\left(Q^*(s,a) - V^*(s)\right)\right), \tag{14}$$

where d^* is the stationary distribution of π^*. To get π^* from w^*, previous policy extraction methods in DICE methods include weighted behavior cloning (Mao et al., 2024b), information projection (Lee et al., 2021) or policy gradient (Nachum et al., 2019b). All these methods parametrize an unimodal Gaussian policy in order to compute $\log \pi(a|s)$ (Haarnoja et al., 2018), which greatly limits its expressivity.

3.5 Goal-Conditioned Behavioral Cloning (GCBC)

We'll start with a very simple method for using generative models to get to a goal: *goal-conditioned imitation learning* (Emmons et al., 2022; Ghosh et al., 2021). The predominant strategy for learning goal-directed skills today is via imitation. This is the basis for much of the recent success in robot learning (Black et al., 2024a; Lynch and Sermanet, 2020; Team et al., 2023; Wang et al., 2022). This looks something like the following:

$$\max_\pi \mathbb{E}_{\tau\sim\mathcal{D},(s,a,s_f)}[\log \pi_\theta(a \mid s, g = s_f)].$$

One way to understand what this is doing is via Bayes' Rule (Eysenbach et al., 2022c; Ghugare et al., 2024; Paster et al., 2022): learning a policy that maximizes the likelihood above is equivalent to maximizing the following objective:

$$\begin{aligned} &\max_{\pi_\theta} \ \mathbb{E}_{a\sim\pi_\theta(a|s,g)}[\log p^\pi(s_f = g \mid s,a) + \log \beta(a \mid s) - \cancel{p(g \mid s)}] \\ &= \mathbb{E}_{a\sim\pi_\theta(a|s,g)}\left[\log \mathbb{E}\left[\sum_t \gamma^t \mathbb{1}(s_t = g) \mid s,a\right] + \log \beta(a \mid s)\right], \end{aligned}$$

where $\beta(a|s)$ is the behavior policy. That is, this is equivalent to selecting actions that maximize your expected reward (+1 when you reach the goal) while also maximizing the likelihood of the actions you've seen in your data (this can be interpreted as a sort of regularization).

Masking interpretation. One interpretation of these GCBC methods is that they're doing masking, a'la BERT. They "cover up" some of the state and action tokens in a sequence and train the model to predict those missing states/actions (Carroll et al., 2022; Chen et al., 2021; Janner et al., 2022). One cool thing about this class of methods, besides working well in practice, is that they blur the line between world model and policy: one generative model can both sample actions and next observations.

3.6 Goal-conditioned RL from the dual perspective

After learning about the dual formulation, we present a way to apply this to goal-conditioned problems as a richer learning signal.

We derive an algorithm where the agents learn by minimizing the following f-divergence:

$$J(\theta) = D_f(p_\theta(s)||p_g(s)) \tag{15}$$

Theorem 3.1. *The gradient of $J(\theta)$ as defined in Equation 15 is given by,*

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\Big[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\Big]\Big[\sum_{t=1}^{T} f'\Big(\frac{p_\theta(s_t)}{p_g(s_t)}\Big)\Big]\right]. \tag{16}$$

The gradient looks exactly like policy gradient with rewards $-f'\left(\frac{p_\theta(s_t)}{p_g(s_t)}\right)$. However, this does not mean that we are maximizing $J^{RL}(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\Big[-f'\Big(\frac{p_\theta(s_t)}{p_g(s_t)}\Big)\Big]$. This is because the gradient of $J^{RL}(\theta)$ is not the same as $\nabla_\theta J(\theta)$.

However, one downside of this objective and its connection to policy gradient is that it is an on-policy objective, and therefore more data inefficient.

3.7 Offline Goal-conditioned RL from the Dual Perspective

To address this we look to a mixture distribution matching objective for the offline setting. Define a *goal-transition distribution* $q(s,a,g)$ in a stochastic MDP as $q(s,a,g) \propto q^{\texttt{train}}(g)\mathbb{E}s' \sim p(\cdot \mid s,a)\mathbb{I}_{\phi(s')=g}$. Intuitively, the distribution has probability mass on each transition that leads to a goal. We formulate the GCRL problem as an occupancy matching problem by searching for the policy π_g that minimizes the discrepancy between its state-action-goal occupancy distribution and the goal-transition distribution $q(s,a,g)$:

$$\texttt{Occupancy matching problem: } \mathcal{D}_f(d^{\pi_g}(s,a,g)\|q(s,a,g)), \tag{17}$$

where D_f denotes an f-divergence with generator function f. Note that the q distribution is potentially unachievable by any goal-conditioned policy π_g. Firstly, it does not account for the initial transient phase that the policy must navigate to reach the desired goal. Secondly, even if we consider only the stationary regime (when $\gamma \to 1$), it may not be dynamically possible for the policy to continuously remain at the goal and rather necessitate cycling around the goal.

Consider a stochastic MDP, a stochastic policy π, and a sparse reward function $r(s,a,g) = \mathbb{E}s' \sim p(\cdot|s,a)\mathbb{I}(\phi(s') = g, q^{\texttt{train}}(g) > 0)$ where $\mathbb{I}$ is an indicator function. Define a soft goal transition distribution to be $q(s,a,g) \propto \exp(\alpha\, r(s,a,g))$. The following bounds hold for any f-divergence that upper bounds KL-divergence (eg. χ^2, Jensen-Shannon):

$$J^{train}(\pi_g) + \frac{1}{\alpha}\mathcal{H}(d^{\pi_g}) \geq -\frac{1}{\alpha}\mathcal{D}_f(d^{\pi_g}(s,a,g)\|q(s,a,g)) + C,$$

where $\mathcal{H}$ denotes the entropy, α is a temperature parameter and C is the partition function for $e^{R(s,a,g)}$. Furthermore, the bound is tight when f is the KL-divergence.

To devise a stable learning algorithm we consider the Pearson χ^2 divergence. Pearson χ^2 divergence has been found to lead to distribution matching objectives that are stable to train as a result of a smooth quadratic generator function f (Al-Hafez et al., 2023; Garg et al., 2021; Sikchi et al., 2023). Our dual formulation simplifies to the following objective:

$$\begin{aligned}\max_{\pi_g}\min_S\ & \overbrace{\beta(1-\gamma)\mathbb{E}(s,g)\sim d_0, a\sim\pi_g(\cdot\mid s,g)S(s,a,g) + \beta\gamma\mathbb{E}(s,a,g)\sim q, s'\sim p(\cdot|s,a), a'\sim\pi_g(\cdot\mid s',g)S(s',a',g)}^{\text{Decrease score at transitions under current policy } \pi_g} \\ -\ & \underbrace{\beta\mathbb{E}(s,a,g)\sim qS(s,a,g)}_{\text{Increase score at the proposed goal transition distribution}} + 0.25\underbrace{\mathbb{E}(s,a,g)\sim \texttt{Mix}_\beta(q,\rho)(\gamma S(s',\pi_g(s'),g) - S(s,a,g))^2}_{\text{Smoothness/Bellman regularization}}.\end{aligned} \tag{18}$$

Equation 18 suggests a contrastive procedure, maximizing the score at the goal-transition distribution and minimizing the score at the offline data distribution under the current policy with Bellman regularization. The Bellman regularization has the interpretation of discouraging neighboring S values from deviating far and smoothing the score landscape. Instantiating with KL divergence results in an objective with similar intuition while resembling an InfoNCE (Oord et al., 2018) objective.

It is important to note that S-function is not grounded to any rewards and does not serve as a probability density of reaching goals, but is rather a score function learned via *a Bellman-regularized contrastive learning procedure*.

3.8 But my task isn't described by a goal observation!

While the most conventional way of combining RL and generative models is to replace components in existing RL algorithms (e.g., the world model, the policy) with capable generative models (see Sec. 2, this section has shown how *interaction* forms a generative process. We have discussed how we can sample from this generative process and train this generative process to reach goals – by maximizing the likelihood of sampling a desired goal state.

What about tasks that aren't well described by a single goal state? It would be great if I got back home and my apartment were clean, but "clean" could correspond to many possible observations – I don't care which permutations of books are in my bookshelf. Similarly, I'd like to have a smooth flight back home – the pilot shouldn't just maximize the likelihood of landing in New York, but also should minimize the amount of turbulence encountered enroute.

Can we use this generative AI perspective to solve tasks like this? Can we use this perspective to solve any reward-maximization problem?

4 Part III: Learning likelihoods

In the previous section, we introduced the notion that we can think about *interaction* as a generative process, in the same way that we can think of rendering as a generative process. We have already seen how this perspective leads to new algorithms – specifically, algorithms for reaching certain goals. But this generative-modeling perspective goes beyond goal-reaching; in fact, we'll see that it can be used to solve arbitrary reward-maximization problems. To get there, we'll need to think about the capabilities that generative models provide.

Let's think about goal-conditioned RL (GCRL) from a likelihood perspective, and then we'll see how we can generalize it to fully-general reward maximization problems.

Generative models often provide two key capabilities: sampling, and estimating log-likelihoods. The previous section focused primarily on thinking about interaction as a certain sampling process. But in this section we'll look at the other capability: estimating log-likelihoods. In particular, we'll want to be able to estimate the likelihood that an agent visits a certain state when interacting with an environment. This quantity, which is already well-studied in the RL literature, will provide the key to solving fully-general RL problems.

Roadmap. In this section, we'll start by formally defining this log-likelihood and then discuss how we can estimate this log likelihood, both directly and indirectly. We'll conclude by showing how this likelihood connects to Q-functions, and can be modeled for any policy as a dot product of learned state representations.

4.1 Log-Likelihoods of the Interaction Generative Process

So, you have a policy interacting with an environment. In the previous part, we've noted that this can be viewed as a generative process, where a sample corresponds to one of the observations visited by the policy. To make this 100% precise, the sample is an observation sampled at time step t, where $t \sim \textsc{Geom}(1-\gamma)$ is a geometric random variable. The intuition here is that we're primarily going to look at the observations that the agent sees after a small number of time steps, but will sometimes look at observations that take more time to reach. The intuition behind the geometric distribution is the same as in RL – it is both mathematically convenient (it is the unique

Figure 11: The **successor measure**, also known as the discounted state occupancy measure, is the distribution over future states that the agent might end up in, at some point in the future. Sec. 4 introduces techniques for estimating these distributions, and then using those estimates for RL.

distribution with the *memoryless property*, which enables TD learning) and functionally useful (as avoids the need for specify a precise "when" the agent reaches a state, but nonetheless incentivizes solving tasks quickly). OK, so we know how to generate samples.

But how do we estimate the *likelihood* of a certain sample? That is, how do we estimate the likelihood of "sampling" observation s? How do we know whether this is an observation that the policy is very likely to encounter, or very unlikely to encounter?

Let's start by formally defining this likelihood. Then Sections 4.2 and 4.3 will provide computation tools for estimating the likelihood. The likelihood is exactly the same as the discounted state occupancy measure (also known as the successor measure) defined in Eq. 5:

$$p^\pi(s_f) \triangleq (1-\gamma)\mathbb{E}_\pi\left[\sum_t \gamma^t p(s_{t+1} = g \mid s_t, a_t)\right].$$

Note that the factors of $(1-\gamma)\gamma^t$ exactly correspond to the PMF of the geometric random variable t. One can also define distributions over state-action pairs $p^\pi(s_f, a_f)$ analogously; the algorithms below are easily extensible to that setting.

4.2 Directly Estimating Likelihoods

One class of goal-reaching methods directly estimates the discounted state occupancy measure. That is, we can fit a parametric model $p_\theta(s_f) \approx p^\pi(s_f)$, or some conditional variant of this ($p_\theta(s_f \mid s, a) \approx p^\pi(s_f \mid s, a)$).

Prior work has done this with various choice of generative models, including flow-matching (Farebrother et al., 2025), GANs (Janner et al., 2020), and normalizing flows (Ghugare and Eysenbach, 2025; Schroecker and Isbell, 2020) However, fitting this density directly can be challenging in high-dimensional settings (Ghugare and Eysenbach, 2025; Janner et al., 2020; Schroecker and Isbell, 2020).

Goal-reaching as an application. So, after you learn these likelihoods, what can you do with them? One simple application is goal reaching. The policy can be optimized to select actions that maximize the likelihood of reaching the desired goal state:[2]

$$\arg\max_a p^\pi(s_f = g \mid s, a), \qquad \max_\pi \mathbb{E}_{\pi(a|s,g)}[p^\pi(s_f = g \mid s, a)].$$

For context, there are really two generative models here:

- We're using (say) a normalizing flow to estimate what states will be visited. We're using the log-likelihood from this model to figure out how to select actions.
- The policy interacting with the MDP defines a generative process/model. We are sampling using this model. Note that we cannot directly estimate the likelihood of samples using this model, which is why we're using (say) the normalizing flow to do that.

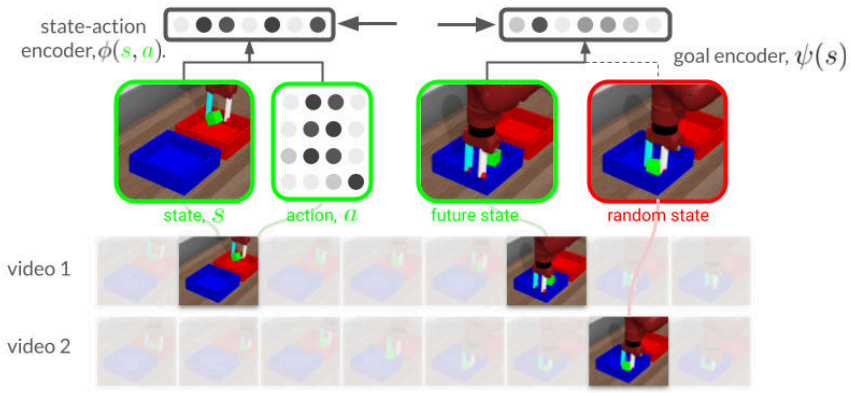


Figure 12: **Estimating Relative Likelihoods via Temporal Contrastive Learning.** (Blier et al., 2021; Eysenbach et al., 2022d; Oord et al., 2018; Sermanet et al., 2018) One way to estimate the successor measure (Fig. 11) is by learning a classifier that distinguishes pairs of observations from the same trajectory from pairs drawn from different trajectories. To make this connection mathematically precise, the positive pairs are offset by a time interval drawn from geometric distribution. Prior work varies in whether actions and in the precise architecture for this classifier.

4.3 Estimating Relative Likelihoods via Temporal Contrastive Learning

An alternative approach, which both sidesteps the difficulties of high-dimensional density estimation, is to estimate the *relative* discounted state occupancy measure:

$$\frac{p^\pi(s_f \mid s, a)}{p(s_f)}.$$

Note that adding the term in the denominator doesn't change the dependence on actions. Namely,

$$\arg\max_a \frac{p^\pi(s_f \mid s, a)}{p(s_f)} = \arg\max_a p^\pi(s_f \mid s, a).$$

The problem of estimating probability ratios is a problem of classification: guessing whether an instance of a random variable came from one distribution or another. Let's see how this works in our setting, where we want to distinguish $p^\pi(s_f \mid s, a)$ from $p(s_f)$.

We will learn the classifier using positive examples $p^\pi(s_f \mid s, a)p(s, a)$ and negative examples $p(s_f)p(s, a)$. The let $f_\theta(s, a, s_f) \in [0, \infty)$ denote the probability ratio of a classifier that takes as input the current state and action and a future state, and guesses whether the future state is a real future state or a random future state. We will train this classifier using the standard binary cross entropy loss:

$$\min_\theta -\mathbb{E}_{p^\pi(s_f|s,a)p(s,a)}\left[\log \frac{1}{1 + \frac{1}{f_\theta(s,a,s_f)}}\right] - \mathbb{E}_{p^\pi(s_f)p(s,a)}\left[\log \frac{1}{1 + f_\theta(s, a, s_f)}\right]. \tag{19}$$

An alternative to the binary cross entropy loss is the Sugiyama least squares importance filtering loss (Blier et al., 2021; Kanamori et al., 2009; Nachum et al., 2019a):

$$\min_\theta \frac{1}{2}\mathbb{E}_{p^\pi(s_f)p(s,a)}[f_\theta(s, a, s_f)^2] - \mathbb{E}_{p^\pi(s_f|s,a)p(s,a)}[f_\theta(s, a, s_f)].$$

Incidentally, this loss keeps being rediscovered, and probably goes back earlier than the 2009 citation above. In both cases, the optimal classifier satisfies:

$$f_\theta(s, a, s_f) = \frac{p^\pi(s_f \mid s, a)}{p(s_f)}.$$

[2]Formally, this conditional occupancy measure is equivalent to the Q function.

Note that this exactly corresponds to temporal contrastive learning (Mobahi et al., 2009; Oord et al., 2018; Sermanet et al., 2018). While we have presented just two losses, there are several others losses that one might choose (Assran et al., 2023; Bardes et al., 2021; Chen and He, 2021; Grill et al., 2020). This has close connections to work in computer vision on temporal coherence (Hénaff et al., 2019; Jayaraman and Grauman, 2016), straightened representations in LLMs (Hosseini and Fedorenko, 2023), and unsupervised learning in neuroscience (Wiskott, 2022).

4.4 Temporal Difference Methods

When thinking about the likelihood of visiting a certain state, that likelihood depends on the policy we're using for selecting actions. The methods we've discussed so far are Monte Carlo methods: they look at where (empirically) the policy visits, and predict that the policy will continue to visit those states. However, there are many settings where one might want to answer a different question: if you changed your policy, what new states would it visit, and what old states would it stop visiting? Classically, this question can be answered with temporal difference methods, which "stitch" together data from different experiences (Ziebart et al., 2008). And it turns out we can do the same thing for these classifier/contrastive methods.

The key idea here is the following identity (Puterman, 2014), sometimes known as the *Bellman flow constraint*, which relates the probability of visiting a state after today to the probability of visiting that state after tomorrow:

$$p^\pi(s_f \mid s_0, a_0) = p(s_1 = s_f \mid s_0, a_0) + \gamma \mathbb{E}_{p(s_1|s_0,a_0)\pi(a_1|s_1)}[p^\pi(s_f \mid s_1, a_1)]. \tag{20}$$

This identity holds for *all* policies, even for ones that are different from that which collected the data. This same identity holds for probability ratios, like those learned in the previous section. We can use this identity to rewrite the previous losses in terms that depend just on *transitions* and the policy – without needing the sample future states:

$$\min_f \mathbb{E}_{p(s,a,s')}[\cdots \log f(s,a,s_f = s') \cdots] + \mathbb{E}_{p(s,a,s')\pi(a'|s')p(s_f)}[f(s',a',s_f) \cdots \log f(s,a,s_f) \cdots].$$

There are several losses that take this form, including the Forward Backward representation (Blier et al., 2021), C-learning (Eysenbach et al., 2021), and TD InfoNCE (Zheng et al., 2024b). The term $f(s',a',s_f)$ acts like an importance weight, which enables us to do off-policy evaluation.[3] This same term $f(s',a',s_f)$ is analogous to the TD target in regular TD learning – in fact, one can show that some versions of this loss result in updates that are isomorphic to value iteration (Eysenbach et al., 2021).

These updates can be done on several policies in parallel. Given a collection of $\{\pi_z(a \mid s) \mid z \sim \mathcal{Z}\}$, we can simultaneously learn $f_z(s,a,s_f)$ by sampling $z \sim \mathcal{Z}$ when doing these TD updates.

4.5 Using these Ratios for Fully General RL Problems

So, we can learn these density ratios. As noted above, these density ratios give us a way of solving goal-reaching tasks: by selecting the action that maximizes the likelihood of reaching a particular goal state. However, certain practical problems of interest are defined by behaviors that are not uniquely identified by one goal. For example, maybe we want to pick up the object in front of us, regardless of what object that is. Maybe we want to navigate to a particular goal on the other side of the room, but do it while making as little sound as possible. Practically, in some settings you have not just one desired goal state, but many. Or, maybe you have some examples of good outcomes and some examples of bad outcomes. Or, maybe you have a set of states labeled with rewards (i.e., the fully-general RL problems). We'll see how these same estimated probabilities can be used to address any of these problems.

While the math above focused on the probability of reaching a single goal, we can use those same estimated probabilities to estimate the likelihoods of other sorts of events: multiple good outcomes, bad outcomes, outcomes with varying levels of goodness (i.e., rewards).

[3] This is another argument for why your TD loss shouldn't backpropagate gradients through your target network (Baird et al., 1995) – because it's acting like an importance weight.

Let's see how this works. Let's say that we have a set of reward-labeled states $\{(s, r(s)\}$, and let's assume that the distribution over these states is the same as the $p(s_f)$ distribution used for learning the probability ratio. For simplicity, we'll assume that the rewards depend only on the current state. Then, after learning the representations, we can estimate the future expected rewards as (Hatch et al., 2023; Mazoure et al., 2023; Touati and Ollivier, 2021):

$$\begin{aligned} Q(s,a) &= \mathbb{E}_{p(s_f|s,a)}[r(s)] \\ &= \mathbb{E}_{p(s_f)}\left[\frac{p(s_f \mid s,a)}{p(s_f)} r(s)\right] \\ &\approx \mathbb{E}_{p(s_f)}\left[f_\theta(s,a,s_f) r(s)\right] \\ &\approx \frac{1}{k}\sum_{s_f,r} f(s,a,s_f) r(s). \end{aligned} \tag{21}$$

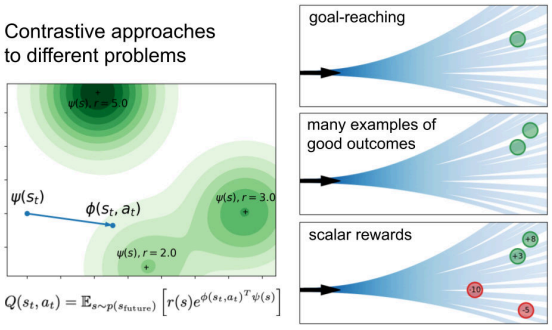


Figure 13: Estimating the probability of future events lets you solve many different types of control problems.

Note that this last expression looks like kernel smoothing. However, whereas kernel methods typically *assume* that the data are structured such that the kernel makes sense, we're learning the representations so that the kernel makes sense.

4.6 Parametrizing the Successor Measure

We previously had thought about the successor measure $f_\theta(s,a,s_f)$ as a black box:

$$f_\theta(s,a,s_f) = NN(s,a,s_f)$$

However, parametrizing the successor measure in terms of representations can improve efficiency and reveals structure in learned representations.

We'll start by looking at two parametrizations that use representations $F_\theta(s,a)$ and $B_\theta(s_f)$:

$$f_\theta(s,a,s_f) = F_\theta(s,a)^T B_\theta(s_f), \qquad f_\theta(s,a,s_f) = e^{F_\theta(s,a)^T B_\theta(s_f)}.$$

We can then think about how learning the successor measure (Sec. 4.3) affects the representations. The first term in Eq. 19 pulls together representations of state-action pairs with real future states. The second term pushes away representations of states that occur in different trajectories. Thus, the resulting representation space is one where states that occur nearby in time have similar representations. If it's difficult to navigate from one state to another, then these two states have distant representations.

To **select actions**, we will choose the action that *moves* the representation $F_\theta(s,a)$ closest towards the goal representation. In control, we often talk about greedy planning (Thrun, 2002): naïvely taking actions that are pointed at the goal. This strategy can be myopic, failing in settings where there are obstacles. Greedy planning in the original observation space is a bad idea. However, the results above show that greedy planning in the representation space is a great idea.

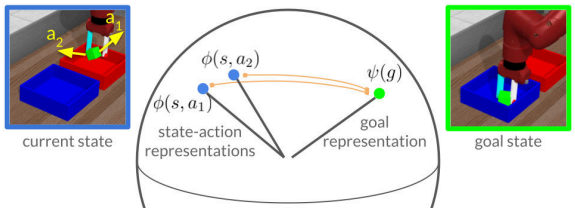


Figure 14: **Action selection.** Once learned, these representations can be used for navigating to goal states: simply select the action whose state-action representation is most similar to the goal representation.

We can also think about these representations as a sort of **model of the world**, one where $\mathcal{F}_\theta(s,a)$ tells you the *representation* of a future state, rather than what the raw future state will look like.

There is a close connection between representations learned by **temporal contrastive learning** and the successor representation (Barreto et al., 2017; Dayan, 1993). In particular, the temporal contrastive representations form a low-rank approximation of the successor representation. The SR and generalizations thereof have been appealing to both computer scientists (Barreto et al., 2018, 2017; Machado et al., 2020) and neuroscientists (Gershman, 2018; Kahn et al., 2024; Momennejad et al., 2017) for its capacity to explain generalization – once learned the SR can be used to quickly estimate the optimal actions for downstream reward functions.

Linear (Agarwal et al., 2025; Blier et al., 2021; Touati et al., 2023)

$$f_\theta(s, a, s_f) = F_\theta(s, a)^T B(s_f).$$

One cool thing about this parametrization is that the Q function (Eq. 21) is a linear function of the representations:

$$\begin{aligned} Q(s, a) &\approx \frac{1}{k} \sum_{s_f, r} f(s, a, s_f) r \\ &= \frac{1}{k} \sum_{s_f, r} F_\theta(s, a)^T B_\theta(s_f) \cdot r \\ &= F_\theta(s, a)^T \underbrace{\left(\frac{1}{k} \sum_{s_f, r} B_\theta(s_f) \cdot r \right)}_{z}. \end{aligned}$$

Thus, we can think about representing a reward function as a single vector z.

Log-Linear (Eysenbach et al., 2022d; Oord et al., 2018)

$$f_\theta(s, a, s_f) = e^{F_\theta(s,a)^T B_\theta(s_f)}.$$

This parameterization lends itself nicely to the classification methods described in Section 4.3 and has ties to temporal contrastive methods (Sermanet et al., 2018).

Cool properties of these log-linear representations include:

1. Unlock scaling (Wang et al., 2025). In most RL settings, bigger networks yield only marginally improved results; we typically don't see the same sort of phrase-transition w.r.t. network depth that we've seen in other domains. However, with these self-supervised approaches, we do start to see this, with significantly increased performance up to 1000-layer networks.
2. Emergent exploration (Liu et al., 2025). We find that we can keep asking the agent to go to the same (difficult) goal over and over, and exploration strategies emerge throughout the course of learning. This also happens in multi-agent settings (Nimonkar et al., 2025).
3. These representations are effective for solving combinatorial problems, such as the Rubik's cube and Sokoban (Ziarko et al., 2025).
4. These representations obey a certain triangle inequality, and you can get TD-like method for "free" but using appropriate neural network architectures (Myers et al., 2024). One intriguing consequence of this is that these representations enable *horizon generalization* – train on short-horizon tasks, and you'll be able to solve long-horizon tasks (Myers et al., 2025).
5. Planning via interpolation (Eysenbach et al., 2024; Zheng et al., 2024a). Given the intuition developed above, one might guess that simply drawing a line from one representation to another provides a way of doing planning. Perhaps surprisingly, this works well in practice and is theoretically justified.

Proto Successor Measure The downside of these above parametrization is that the representation it learns is conditioned on the task, in effect it learns a separate representation for each task z. Instead, we propose to learn a new breakdown of successor measures using the following observation: Any successor measure, M^π in an MDP forms an affine set and so can be represented as $\sum_i^d \phi_i w_i^\pi + b$ where ϕ_i and b are independent of the policy π and d is the dimension of the affine space.

$$m(s, a, s^+, a^+) = \Phi(s, a, s^+, a^+)^\top w^\pi + b(s, a, s^+, a^+)$$

This is the basis for proto successor measure (PSM) which learns the basis set for all possible behaviors.

For a given policy π, its successor measure under our framework is denoted by $M^\pi = \Phi w^\pi + b$ with w^π the only object depending on policy. Given an offline dataset with density ρ, we follow prior works (Blier et al., 2021; Touati and Ollivier, 2021) and model densities $m^\pi = M^\pi/\rho$ learned with the following objective:

$$\begin{aligned} L^\pi(\Phi, b, w^\pi) = -(1-\gamma)\mathbb{E}_{s,a\sim\rho}[m^{\Phi,b,w^\pi}(s, a, s, a)] \\ + \frac{1}{2}\mathbb{E}_{s,a,s'\sim\rho,s^+,a^+\sim\rho}[(m^{\Phi,b,w^\pi}(s, a, s^+, a^+) - \\ \gamma \bar{m}^{\bar{\Phi},\bar{b},\bar{w}^\pi}(s', \pi(s'), s^+, a^+))^2]. \end{aligned} \quad (22)$$

The above objective only requires samples (s, a, s') from the reward-free dataset and a random state-action pair (s^+, a^+) (also sampled from the same data) to compute $L(\pi)$. Without loss of generality we can combine the bias in the basis, giving us the parameterization:

$$M^\pi(s, a, s^+) = \psi^\pi(s, a)\varphi(s^+)\rho(s^+).$$

4.7 What can we do with Zero-shot RL Methods: RLZero

Now we have a method that gives us the optimal policy for any reward function. What can we do with this? It's not always easy to specify tasks through rewards. What else can we do?

We can use these zero-shot RL methods – without task-specific supervision or labeled trajectories – to get zero-shot test-time policy inference from arbitrary natural language instructions (Sikchi et al., 2025). We introduce a framework comprising three steps: *imagine*, *project*, and *imitate*. First, the agent imagines a sequence of observations corresponding to the provided language description using video generative models. Next, these imagined observations are projected into the target environment domain. Finally, an agent pretrained in the target environment with unsupervised RL instantly imitates the projected observation sequence through a closed-form solution. RLZero demonstrates direct language-to-behavior generation abilities on a variety of tasks and environments without any in-domain supervision. Components of RLZero can also be used to generate policies zero-shot from cross-embodied videos, such as those available on YouTube, even for complex embodiments like humanoids.

Transition. In most real-world settings we have no data, or only partial data. So, we have to explore the environment and collect more data, to learn about our surroundings? How do we do this? Note that this is fundamentally a question left unanswered by typical generative models, but it's one where RL tools are nicely aligned.

5 Part IV: Self-Generated Feedback

Where have we been? At the end of Part 1, we discussed how we can view RL as a joint optimization problem over data and policies. We discussed connections with dual RL. And we intuitively said that, if you didn't have a simulator, you might try to constrain your optimized data to be close to data you've seen before. But, the reward function provided a bit of a scaffold, on which data stands. Part 2 removed part of this crutch. While rewards often specify not just *what* a

task is but also *how* a task should be solved, the goal-conditioned setting allows users to specify the what without the how. This, combined with the use of simulation, enabled algorithms that could collect their own data. Part 3 lifted a limitation of these goal-reaching methods, showing that the probabilistic/generative perspective on RL not only gives us new tools for solving goal-reaching problems, but also enables us to solve fully-general RL problems.

How can RL enable generative AI systems to construct their own knowledge? Now, in this final part, I want to return to the role of data. So far, we've been thinking about data as a means to an end – how do you find the data that "supports" your reward-maximizing policy. But how can you discover data on your own? How can we grow a brain from scratch? How can we build generative models that construct their own datasets?

5.1 The exploration problem

Perhaps the most common way of thinking about exploration is in terms of coverage – we want the agent to explore as many states as possible (Bellemare et al., 2016; Hazan et al., 2019; Lee et al., 2019; Ostrovski et al., 2017; Strehl and Littman, 2008). We can use the same lens of occupancy measures (distributions over states) to think about exploration – rather than learning an occupancy measure that assigns high probability to high-reward trajectories, you want to learn an occupancy measure that assigns some probability to every trajectory.

However, exploration is about more than going everywhere – it's also about preparing to solve new tasks. How can we put the agent in a position so that it can quickly solve new tasks?

Figure 15: **Empowerment** drives agents to go to states where they can enact change. Most immediately, this will be states that are centrally-located, like a fire station in the middle of a city. Maximizing empowerment will also lead to the accumulation of resources (capital), friends (social capital), influence (political capital), and knowledge.

5.2 Empowerment: put yourself in a place where you can do many things

Empowerment (Klyubin et al., 2005; Salge et al., 2014) is the usual language to think about this, well studied in psychology. Empowerment is usually thought about as an agent navigating to a position in state space where it can do many things. But now we're thinking about the exploration problem as one of empowerment: the "state" of the agent is both its physical location in the environment, but also its state-of-mind, its knowledge about the world (as reflected in the data collected so far).

Early work on empowerment (Klyubin et al., 2005) looks at the objective $I(s_t; a_{0:t})$, which is the mutual information between the state of the world at time t and a sequence of actions $a_{0:t}$. Maximizing this MI corresponds to selecting a distribution over action sequences so that s_t takes on a large number of values (i.e., exploration), but each particular sequence results in a narrow set of possible future states s_t. Compared with the objectives we have been looking at so far, this objective is *open-loop*, in the sense that actions are chosen without reference to the current state.

Mohamed and Jimenez Rezende (2015) use an objective that looks like:

$$\begin{aligned} I(a; s' \mid s) &= \mathcal{H}[s' \mid s] - \mathcal{H}[s' \mid s, a] \\ &= \mathcal{H}[a \mid s] - \mathcal{H}[a \mid s, s']. \end{aligned}$$

Intuitively, this means that we want to select actions that exert a high degree of influence over the next state s'. And we're selecting these actions using information about the current state. Note that we're treating this like an RL problem, so our goal is to maximize this objective both now and in the future; the optimal policy will try to navigate to states where actions exert a high degree of influence over future outcomes.

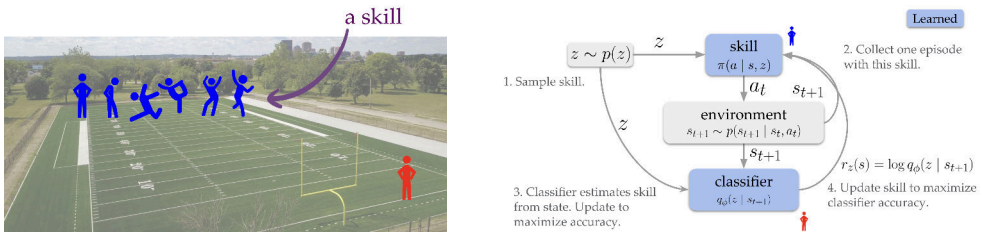


Figure 16: **Unsupervised skill learning.** *(Left)* Skill learning can be viewed as a game, where one player (blue) who performs actions to communicate a word to another player (red) who guesses the word from the actions they see. *(Right)* Overview of how this game translates into a practical algorithm.

There is also cool work extending empowerment to multi-agent settings – rather than having an agent try to empower itself, we can have an AI agent try to empower a human agent (Du et al., 2020; Myers et al., 2024).

Summary. But from a computational perspective, pure exploration or empowerment maximization doesn't equip us with the skills to go on and do many different things. How do we do this? The key idea is to replace action with a learned temporal abstraction, which we'll call a "skill". So, one way of interpreting skill learning algorithms is that they are maximizing empowerment, but doing it over *learned* temporally-abstracted actions.

5.3 Skills

Our goal is to not just explore an environment, but learn *skills* that will enable us to rapidly solve downstream tasks (see Fig. 17). This area is sometimes known as *self-supervised reinforcement learning* – a family of methods in which agents autonomously generate their own rewards (e.g., via intrinsic motivation) to learn skills that cover this large space of behaviors. These skills are not programmed in advance, but rather are discovered through exploring and experimenting, without using human demonstrations. By efficiently representing the space of possible behaviors, this set of skills (sometimes called a *behavioral foundation model*) can be leveraged to rapidly solve new tasks.

Intuition as a game (VISR (Warde-Farley et al., 2018)). Imagine playing a game that has two players, who stand on either end of a football field (see Fig. 16 (left)). One player (blue) is trying to communicate a word. The other player (red) is trying to guess the word from the behavior performed by the first player. This is a cooperative game. The blue player will do one behavior when they want to convey one word, and a different behavior when they want to convey a different word. At the end of each round, the red and blue players talk amongst themselves, so that the red player can get feedback (what was the blue player *actually* trying to say) and blue player can get feedback (what behavior *should* I have done to convey the word "dog").

How would you play this game? Would you make small behaviors or big behaviors? Similar behaviors or pretty distinct ones?

This game maps exactly onto notions from coding theory and channel capacity. In fact, this connection provides leverage to understand the sense in which this is optimal (see Fig. 17 (right)).

An objective for skill learning: Intuition. There are several metrics one might use for quantifying whether a set of skills is good or not. One natural metric is coverage: Does the set of skills collectively explore and visit a wide range of states? This metric is well-established in prior work on exploration (Hazan et al., 2019; Ostrovski et al., 2017).

Another metric is uniqueness: does each skill perform behavior that is distinct from other skills? This metric is important not just from an efficiency perspective (it would be redundant to have two skills that perform similar behaviors) but also from the perspective of solving downstream tasks: if two skills are highly similar, it can be more challenging to figure out which should be used to

solve a particular downstream task. The combination of the coverage metric and this uniqueness metric is well-established in the skill learning literature (Achiam et al., 2018; Co-Reyes et al., 2018).

An objective for skill learning: Math. OK, so let's translate this into math to think about how this unsupervised pretraining will work. When we're talking about skills, we'll use $p^\pi(s_f \mid s, a, z)$ to denote the distribution corresponding to skill $\pi(a \mid s, z)$

The goal of unsupervised pretraining is to learn a skill-conditioned policy $\pi : \mathcal{S} \times \mathcal{Z} \mapsto \Delta(\mathcal{A})$ that conducts diverse and discriminable behaviors, where $\mathcal{Z}$ is a latent skill space. We will use $p(z)$ to denote the prior distribution over skills. Typically, this is a uniform categorical distribution (i.e., z is a random integer from $1 \cdots N$, typically encoded as a one-hot vector), a Gaussian distribution, or a uniform distribution over the d-dimensional unit hypersphere $p(z) = \textsc{Unif}(\mathrm{S}^{d-1})$.

Our goal is to learn policies so that input z is a "knob" for sweeping over the space of behaviors.

To learn skills, we're going to think about the mutual information (MI) between the random variable z and the behavior induced. We'll use $\tau = (s_0, a_0, \cdots)$ to denote a trajectory of experience. Thus, the quantity that we're maximizing is

$$\max_\pi I^\pi(z; \tau).$$

One way of understanding this objective is that we want z to exert a high degree of influence over the behavior τ.

Another way of understanding this objective is by writing the MI in terms of entropy:

$$I^\pi(z; \tau) = \mathcal{H}[\tau] - \mathcal{H}[\tau \mid z].$$

The first term on the RHS, $\mathcal{H}[\tau]$, is the marginal entropy over behaviors. Thus, maximizing MI corresponds to trying to produce the broadest possible distribution over behaviors. This term corresponds to pure exploration. The second term on the RHS, $\mathcal{H}[\tau \mid z]$, minimizes the *conditional* entropy over behaviors; for a particular skill z, you want the policy to have a *predictable* behavior. Predictability is important because we want to be able to use skills to solve downstream tasks. For solving downstream tasks, it is important to be able to predict what, say, skill $z = 1$ will do. If it sometimes goes left and sometimes goes right, then it's much harder to use this skill to solve more complex tasks.

A Practical Algorithm There are lots of different methods for doing skill learning. They typically fit a similar mold (see Fig. 16 (right)):

- **Policy optimization.** The inner expectation is equivalent to an RL problem with the reward function $r_t = \log q(z \mid s_t)$:

  $$\mathbb{E}_{p^\pi(s_f|z)}[\log q(z \mid s_t)] = (1-\gamma)\mathbb{E}_{\tau \sim \pi(\cdot|\cdot,z)}[\sum_t \gamma^t \log q(z \mid s_t)].$$

  Thus, given a fixed q, we can optimize the objective w.r.t. π using any off-the-shelf RL algorithm, using the learned q as the reward function. This could be DQN, TD3, PPO, etc.
- **Posterior optimization.** The optimization problem w.r.t. q is a standard maximum likelihood problem. We sample training data (s, z) as $z \sim p(z), s \sim \rho^\pi(s \mid z)$, and then do maximum likelihood to maximize $\log q(z \mid s)$.

  A number of papers assume that z is a normalized vector, so $q(z \mid s)$ takes the form of a von-Mises Fisher distribution: $q(z \mid s) = \frac{1}{Z} e^{z^T \phi(s)}$ (Hansen et al., 2019; Park et al., 2023; Warde-Farley et al., 2018; Zheng et al., 2025).

Solving downstream tasks There are many ways that skills can be used to solve downstream tasks.

One option (Eysenbach et al., 2019; Hansen et al., 2019; He et al., 2022; Park et al., 2023) is to treat the set of skills as a small hypothesis space of policies. We identify the skill that best solves a downstream task. Because skills are intended to be diverse, the expectation is that we haven't incurred much error from shrinking the hypothesis space in this way. One important question

here is how this skill is identified. If the number of skills is small, we can simply **enumerate** all the skills. For example, if someone gives us a reward function, we can simply measure which skill gets the highest reward, and use that skill. Another option is to use the posterior $q(z \mid \tau)$ to **infer** the best skill. This approach requires that we observe a trajectory or states from the behavior that we want to emulate. This approach shows how skills provide one approach to zero-shot RL (Sikchi et al., 2025; Touati et al., 2023).

Another approach (Co-Reyes et al., 2018; Eysenbach et al., 2019; Florensa et al., 2017; Gregor et al., 2016; Sharma et al., 2019) is to treat the skills as high-level, temporally-extended actions. They define a new MDP, where actions correspond to skills and where the horizon is substantially shorter. This new MDP can be solved with any off-the-shelf RL algorithm. Some prior work has also used the learned posterior to facilitate this (Sharma et al., 2019). This is akin to hierarchical RL, but where the low-level policies are learned with a different objective (MI, not reward maximization). This can be appealing because learning low-level skills in a hierarchical framework can be unstable.

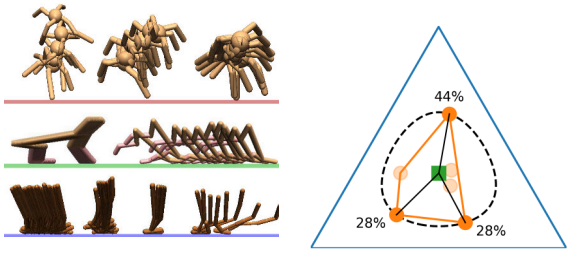


Figure 17: Unsupervised skill learning explores the world, acquiring skills that enable the agent to quickly solve new tasks. One can use information geometry to prove how and when these skills are optimal (Eysenbach et al., 2022b).

5.4 Compression Perspective

I now want to zoom out and think more holistically about what these skill learning methods are providing. Fundamentally, ML is a problem of compression. Standard algorithms for supervised learning, such as classification and regression, are compressing Y given X. Their objectives can be formally described as conveying the number of bits you need to send to convey the labels, *assuming the recipient already knows the inputs* X. Unsupervised learning aims to compress a set of inputs X. This is what a VAE does, this is what LLM pretraining does – it's all about compressing your given dataset.

What self-supervised RL does fundamentally different. In unsupervised/self-supervised RL, we compress an *MDP*. That is, the object that we're compressing isn't data because, well, we aren't given any data to compress. Rather, we compress an MDP\R, an MDP without a reward function. The artifact that is produced by this compression is a behavioral foundation model, a policy of some sort. In the same way that image generative models and LLMs can be prompted, this model is prompt-able.

But there's a key difference here – whereas typical generative models (audio, speech, language, videos) are trained with curated data, these RL methods can be trained on self-collected data. Thus, the question they are answering is different – it's not, "factor the training data and produce similar things," but rather it is "explore the environment and discover interesting things." This capacity has the potential to lift fundamental challenges with AI today (where does our data come from) and open up new and important applications of AI.

6 Conclusion

So, where have we been today? We started in Part I by thinking about how generative models can be plugged into existing RL algorithms, whether as policies or world models. Part II introduced the notion that interaction itself defines a generative process, and that thinking of interaction as a generative model gave us new algorithms for learning policies. Part III built upon this to think about estimating the likelihoods of this generative process, which ultimately gave us tools for solving fully-general RL problems. Finally, Part IV briefly discussed the provenance of data – building generative models that collect their own data.

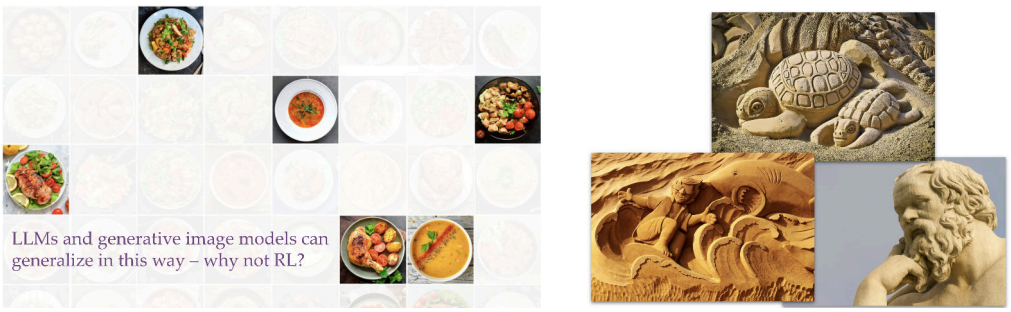


Figure 19: **Generalization is the next frontier.** In the same way that generative image models can produce life-like images after only training on a small fraction of possible images, how might we build (generative) RL agents that can *build* (with shovels and trowels) life-like sandcastles after only practicing building a small fraction of possible sandcastles?

CFP on dual RL. As mentioned, a fundamental limitation of the RL framework is the reliance on reward, especially a dense reward. However, most real world problems are often very difficult to design a shaped reward function for, requiring expert domain knowledge and some idea of what the optimal policy is in the first place. However, the draw of reinforcement learning is its ability to improve upon itself to find solutions that no human would have thought of. We must think carefully about the learning signals we can leverage to achieve this.

Dangers of not thinking about interaction. There are a lot of decision-making agents that were trained with bandit or supervised learning methods plugged into real world systems today that are sequential decision-making problems. These myopic solutions to true reinforcement learning problems are suboptimal and unstable (Shumailov et al., 2024). This is true for a variety of problems, not limited to generative modeling applications, such as recommendation systems, educational technology, industrial control, and dialogue systems.

Figure 18: Despite the rapid progress in RL over the last decade, today's algorithms may nonetheless retain only a tiny fraction of the potential capabilities of RL algorithms.

One missing ingredient: generalization. In the last few minutes, I'd like to advocate for rethinking the role of generalization (see Fig. 19). Let's think about the simplest machine learning model – the nearest neighbor model. Given, say, an image, find the closest labeled image you've seen before, and output the same label. We wouldn't say that this nearest neighbor model is smart. What really "wowed" people about deep learning models is that they could make predictions on inputs that looked different than those seen during training. Representations (those trained end to end) were important for facilitating this. What really "wowed" people about generative image and language models over the last couple years was similarly their capacity to produce outputs that are quite different from those seen in the training data.

What does this mean in the context of interactive generative models? We're used to seeing plots showing that agents (typically simulated robots) cover the state space (typically a maze) throughout the course of training. Their high state coverage is cited as a metric for success. The fact that these skills then, at test time, navigate to different parts of the state space is somewhat a given – they've already memorized this during training. But in many practical problems, especially those involving interacting with the world, it is impossible to cover even a tiny slice of the world during training. Any coverage metric will be zero. However, this doesn't mean that learned skills will not be useful for solving downstream tasks. There are 1e17 paths across Manhattan. You can probably navigate them effectively, even though you've only taken a vanishingly small fraction

of them. The key concept is generalization – being able to accomplish a thing doesn't mean that you've necessarily tried it before.

Note that close connection with compression here. In compression, we're trying to find patterns in the data. Identifying these patterns (e.g., cats of whiskers) allows ML models to generalize, to continue to make good predictions (e.g., this image looks like a cat) on unseen examples. The same is true for reinforcement learning. One way of seeing this is by supposing that the agent has a huge number of skills baked into it, each with a certain name (e.g., "bake a yellow cake with orange frosting, 3 layers tall"); of course, the names are arbitrary and just represented as random vectors, but I think the analogy is useful. Most of these skills are never practiced. But, practicing one skill will improve many other skills. So, when practicing skills, we should be thinking not "let's get good at this particular skill", but rather "can I learn key concepts that enable performing many skills later." This is akin to a high-school student learning the key concepts of calculus so that they can solve a wide range of math problems, rather than just memorizing the answers to a select few problems.

The capacity to explore the space of behaviors, and organize it, remains a fundamental open problem in RL. And, I think, AI. Will enable RL methods to tackle problems demanding long-horizon reasoning, problems that humans don't know how to solve. *While today's generative models just paint pixels and tokens on a screen, tomorrow's might be able to build the world that today's generative models can only dream of.*

References

Achiam, J., Edwards, H., Amodei, D., and Abbeel, P. (2018). Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*.

Agarwal, S., Sikchi, H., Stone, P., and Zhang, A. (2025). Proto successor measure: Representing the behavior space of an RL agent. In *Forty-second International Conference on Machine Learning*.

Al-Hafez, F., Tateo, D., Arenz, O., Zhao, G., and Peters, J. (2023). LS-IQ: Implicit reward regularization for inverse reinforcement learning. In *The Eleventh International Conference on Learning Representations*.

Assran, M., Duval, Q., Misra, I., Bojanowski, P., Vincent, P., Rabbat, M., LeCun, Y., and Ballas, N. (2023). Self-supervised learning from images with a joint-embedding predictive architecture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15619–15629.

Baird, L. et al. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the twelfth international conference on machine learning*, pages 30–37.

Baker, B., Akkaya, I., Zhokov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., and Clune, J. (2022). Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654.

Bardes, A., Ponce, J., and LeCun, Y. (2021). Vicreg: Variance-invariance-covariance regularization for self-supervised learning. *arXiv preprint arXiv:2105.04906*.

Barreto, A., Borsa, D., Quan, J., Schaul, T., Silver, D., Hessel, M., Mankowitz, D., Zidek, A., and Munos, R. (2018). Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *International Conference on Machine Learning*, pages 501–510. PMLR.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. (2017). Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30.

Beeler, C., Subramanian, S. G., Bellinger, C., Crowley, M., and Tamblyn, I. (2023). ChemgymRL: An interactive framework for reinforcement learning for digital chemistry. In *NeurIPS 2023 AI for Science Workshop*.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29.

Black, K., Brown, N., Driess, D., Esmail, A., Equi, M., Finn, C., Fusai, N., Groom, L., Hausman, K., Ichter, B., et al. (2024a). π0: A vision-language-action flow model for general robot control. *URL https://arxiv.org/abs/2410.24164*.

Black, K., Janner, M., Du, Y., Kostrikov, I., and Levine, S. (2024b). Training diffusion models with reinforcement learning. In *The Twelfth International Conference on Learning Representations*.

Blier, L., Tallec, C., and Ollivier, Y. (2021). Learning successor states and goal-dependent values: A mathematical viewpoint. *CoRR*, abs/2101.07123.

Boyd, S., Boyd, S. P., and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Carroll, M., Paradise, O., Lin, J., Georgescu, R., Sun, M., Bignell, D., Milani, S., Hofmann, K., Hausknecht, M., Dragan, A., et al. (2022). Uni [mask]: Unified inference in sequential decision problems. *Advances in neural information processing systems*, 35:35365–35378.

Chen, H., Lu, C., Ying, C., Su, H., and Zhu, J. (2022). Offline reinforcement learning via high-fidelity generative behavior modeling. *arXiv preprint arXiv:2209.14548*.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. (2021). Decision transformer: Reinforcement learning via sequence modeling. *Proc. of NeurIPS*.

Chen, X. and He, K. (2021). Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15750–15758.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31.

Co-Reyes, J., Liu, Y., Gupta, A., Eysenbach, B., Abbeel, P., and Levine, S. (2018). Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *International conference on machine learning*, pages 1009–1018. PMLR.

Cranmer, K., Brehmer, J., and Louppe, G. (2020). The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062.

Csilléry, K., Blum, M. G., Gaggiotti, O. E., and François, O. (2010). Approximate bayesian computation (abc) in practice. *Trends in ecology & evolution*, 25(7):410–418.

Dax, M., Green, S. R., Gair, J., Macke, J. H., Buonanno, A., and Schölkopf, B. (2021). Real-time gravitational wave science with neural posterior estimation. *Physical review letters*, 127(24):241103.

Dayan, P. (1993). Improving generalization for temporal difference learning: The successor representation. *Neural computation*, 5(4):613–624.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472.

Du, Y., Tiomkin, S., Kiciman, E., Polani, D., Abbeel, P., and Dragan, A. (2020). Ave: Assistance via empowerment. *Advances in Neural Information Processing Systems*, 33:4560–4571.

Emmons, S., Eysenbach, B., Kostrikov, I., and Levine, S. (2022). Rvs: What is essential for offline RL via supervised learning? In *International Conference on Learning Representations*.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. (2019). Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*.

Eysenbach, B., Khazatsky, A., Levine, S., and Salakhutdinov, R. R. (2022a). Mismatched no more: Joint model-policy optimization for model-based rl. *Advances in Neural Information Processing Systems*, 35:23230–23243.

Eysenbach, B., Myers, V., Salakhutdinov, R., and Levine, S. (2024). Inference via interpolation: Contrastive representations provably enable planning and inference. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Eysenbach, B., Salakhutdinov, R., and Levine, S. (2021). C-learning: Learning to achieve goals via recursive classification. In *International Conference on Learning Representations*.

Eysenbach, B., Salakhutdinov, R., and Levine, S. (2022b). The information geometry of unsupervised reinforcement learning. In *International Conference on Learning Representations*.

Eysenbach, B., Udatha, S., Salakhutdinov, R. R., and Levine, S. (2022c). Imitating past successes can be very suboptimal. *Advances in Neural Information Processing Systems*, 35:6047–6059.

Eysenbach, B., Zhang, T., Levine, S., and Salakhutdinov, R. R. (2022d). Contrastive learning as goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems*, 35:35603–35620.

Farahmand, A.-M., Barreto, A., and Nikovski, D. (2017). Value-Aware Loss Function for Model-based Reinforcement Learning. In Singh, A. and Zhu, J., editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1486–1494. PMLR.

Farebrother, J., Pirotta, M., Tirinzoni, A., Munos, R., Lazaric, A., and Touati, A. (2025). Temporal difference flows. *arXiv preprint arXiv:2503.09817*.

Florensa, C., Duan, Y., and Abbeel, P. (2017). Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*.

Fujimoto, S., Meger, D., and Precup, D. (2019). Off-policy deep reinforcement learning without exploration. In *Proc. of ICML*, pages 2052–2062.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *Proc. of ICML*, pages 1582–1591.

Garg, D., Chakraborty, S., Cundy, C., Song, J., and Ermon, S. (2021). Iq-learn: Inverse soft-q learning for imitation. *Advances in Neural Information Processing Systems*, 34:4028–4039.

Gershman, S. J. (2018). The successor representation: its computational logic and neural substrates. *Journal of Neuroscience*, 38(33):7193–7200.

Ghosh, D., Gupta, A., Reddy, A., Fu, J., Devin, C. M., Eysenbach, B., and Levine, S. (2021). Learning to reach goals via iterated supervised learning. In *Proc. of ICLR*.

Ghugare, R., Bharadhwaj, H., Eysenbach, B., Levine, S., and Salakhutdinov, R. (2023). Simplifying model-based RL: Learning representations, latent-space models, and policies with one objective. In *The Eleventh International Conference on Learning Representations*.

Ghugare, R. and Eysenbach, B. (2025). Normalizing flows are capable models for rl. *arXiv preprint arXiv:2505.23527*.

Ghugare, R., Geist, M., Berseth, G., and Eysenbach, B. (2024). Closing the gap between TD learning and supervised learning - a generalisation point of view. In *The Twelfth International Conference on Learning Representations*.

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

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M., et al. (2020). Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284.

Grimm, C., Barreto, A., Singh, S., and Silver, D. (2020). The value equivalence principle for model-based reinforcement learning. *Advances in neural information processing systems*, 33:5541–5552.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. (2019). Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*.

Hansen, S., Dabney, W., Barreto, A., Van de Wiele, T., Warde-Farley, D., and Mnih, V. (2019). Fast task inference with variational intrinsic successor features. *arXiv preprint arXiv:1906.05030*.

Hansen-Estruch, P., Kostrikov, I., Janner, M., Kuba, J. G., and Levine, S. (2023). Idql: Implicit q-learning as an actor-critic method with diffusion policies. *arXiv preprint arXiv:2304.10573*.

Hatch, K. B., Eysenbach, B., Rafailov, R., Yu, T., Salakhutdinov, R., Levine, S., and Finn, C. (2023). Contrastive example-based control. In *Learning for Dynamics and Control Conference*, pages 155–169. PMLR.

Hazan, E., Kakade, S., Singh, K., and Van Soest, A. (2019). Provably efficient maximum entropy exploration. In *International Conference on Machine Learning*, pages 2681–2691. PMLR.

He, S., Jiang, Y., Zhang, H., Shao, J., and Ji, X. (2022). Wasserstein unsupervised reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6884–6892.

Hénaff, O. J., Goris, R. L., and Simoncelli, E. P. (2019). Perceptual straightening of natural videos. *Nature neuroscience*, 22(6):984–991.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851.

Hosseini, E. and Fedorenko, E. (2023). Large language models implicitly learn to straighten neural sentence trajectories to construct a predictive representation of natural language. *Advances in Neural Information Processing Systems*, 36:43918–43930.

Janner, M., Du, Y., Tenenbaum, J., and Levine, S. (2022). Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*.

Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32.

Janner, M., Li, Q., and Levine, S. (2021). Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*.

Janner, M., Mordatch, I., and Levine, S. (2020). gamma-models: Generative temporal difference learning for infinite-horizon prediction. *Advances in Neural Information Processing Systems*, 33:1724–1735.

Jayaraman, D. and Grauman, K. (2016). Slow and steady feature analysis: higher order temporal coherence in video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3852–3861.

Kaelbling, L. P. (1993). Learning to achieve goals. In *IJCAI*, volume 2, pages 1094–8. Citeseer.

Kahn, A. E., Bassett, D. S., and Daw, N. D. (2024). Trial-by-trial learning of successor representations in human behavior. *bioRxiv*, pages 2024–11.

Kanamori, T., Hido, S., and Sugiyama, M. (2009). A least-squares approach to direct importance estimation. *The Journal of Machine Learning Research*, 10:1391–1445.

Kidambi, R., Rajeswaran, A., Netrapalli, P., and Joachims, T. (2020). Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33:21810–21823.

Kingma, D., Salimans, T., Poole, B., and Ho, J. (2021). Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707.

Klyubin, A. S., Polani, D., and Nehaniv, C. L. (2005). All else being equal be empowered. In *European Conference on Artificial Life*, pages 744–753. Springer.

Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. (2019). Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in neural information processing systems*, 32.

Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial intelligence*, 33(1):1–64.

Lambert, N., Amos, B., Yadan, O., and Calandra, R. (2020). Objective mismatch in model-based reinforcement learning. *arXiv preprint arXiv:2002.04523*.

Lee, J., Jeon, W., Lee, B., Pineau, J., and Kim, K.-E. (2021). Optidice: Offline policy optimization via stationary distribution correction estimation. In *International Conference on Machine Learning*, pages 6120–6130. PMLR.

Lee, L., Eysenbach, B., Parisotto, E., Xing, E., Levine, S., and Salakhutdinov, R. (2019). Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*.

Liu, G., Tang, M., and Eysenbach, B. (2025). A single goal is all you need: Skills and exploration emerge from contrastive RL without rewards, demonstrations, or subgoals. In *The Thirteenth International Conference on Learning Representations*.

Lu, C., Chen, H., Chen, J., Su, H., Li, C., and Zhu, J. (2023). Contrastive energy prediction for exact energy-guided diffusion sampling in offline reinforcement learning. In *International Conference on Machine Learning*, pages 22825–22855. PMLR.

Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., and Zhu, J. (2022). Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787.

Lynch, C. and Sermanet, P. (2020). Language conditioned imitation learning over unstructured data. *arXiv preprint arXiv:2005.07648*.

Machado, M. C., Bellemare, M. G., and Bowling, M. (2020). Count-based exploration with the successor representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5125–5133.

Manne, A. S. (1960). Linear programming and sequential decisions. *Management Science*, 6(3):259–267.

Mao, L., Xu, H., Zhan, X., Zhang, W., and Zhang, A. (2024a). Diffusion-dice: In-sample diffusion guidance for offline reinforcement learning. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS 2024)*, New Orleans, LA, USA.

Mao, L., Xu, H., Zhang, W., and Zhan, X. (2024b). Odice: Revealing the mystery of distribution correction estimation via orthogonal-gradient update. *arXiv preprint arXiv:2402.00348*.

Mazoure, B., Eysenbach, B., Nachum, O., and Tompson, J. (2023). Contrastive value learning: Implicit models for simple offline rl. In *Conference on Robot Learning*, pages 1257–1267. PMLR.

Milani, S., Kanervisto, A., Ramanauskas, K., Schulhoff, S., Houghton, B., and Shah, R. (2024). Bedd: The minerl basalt evaluation and demonstrations dataset for training and benchmarking agents that solve fuzzy tasks. *Advances in Neural Information Processing Systems*, 36.

Mobahi, H., Collobert, R., and Weston, J. (2009). Deep learning from temporal coherence in video. In *Proceedings of the 26th annual international conference on machine learning*, pages 737–744.

Mohamed, S. and Jimenez Rezende, D. (2015). Variational information maximisation for intrinsically motivated reinforcement learning. *Advances in neural information processing systems*, 28.

Momennejad, I., Russek, E. M., Cheong, J. H., Botvinick, M. M., Daw, N. D., and Gershman, S. J. (2017). The successor representation in human reinforcement learning. *Nature human behaviour*, 1(9):680–692.

Myers, V., Ji, C., and Eysenbach, B. (2025). Horizon generalization in reinforcement learning. In *The Thirteenth International Conference on Learning Representations*.

Myers, V., Zheng, C., Dragan, A., Levine, S., and Eysenbach, B. (2024). Learning temporal distances: Contrastive successor features can provide a metric structure for decision-making. In *Forty-first International Conference on Machine Learning*.

Nachum, O., Chow, Y., Dai, B., and Li, L. (2019a). Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. *Advances in neural information processing systems*, 32.

Nachum, O. and Dai, B. (2020). Reinforcement learning via fenchel-rockafellar duality. *arXiv preprint arXiv:2001.01866*.

Nachum, O., Dai, B., Kostrikov, I., Chow, Y., Li, L., and Schuurmans, D. (2019b). Algaedice: Policy gradient from arbitrary experience. *arXiv preprint arXiv:1912.02074*.

Newell, A. and Simon, H. A. (1961). Gps, a program that simulates human thought.

Nimonkar, C., Shah, S., Ji, C., and Eysenbach, B. (2025). Goal-conditioned multi-agent cooperation with contrastive rl. *In submission*.

Oord, A. v. d., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Ostrovski, G., Bellemare, M. G., Oord, A., and Munos, R. (2017). Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. PMLR.

Park, S., Rybkin, O., and Levine, S. (2023). Metra: Scalable unsupervised rl with metric-aware abstraction. *arXiv preprint arXiv:2310.08887*.

Paster, K., McIlraith, S., and Ba, J. (2022). You can't count on luck: Why decision transformers and rvs fail in stochastic environments. *Advances in neural information processing systems*, 35:38966–38979.

Peters, J., Mülling, K., and Altun, Y. (2010). Relative entropy policy search. In *Proc. of AAAI*.

Peters, J. and Schaal, S. (2007). Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th international conference on Machine learning*, pages 745–750.

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings.

Salge, C., Glackin, C., and Polani, D. (2014). Empowerment–an introduction. *Guided Self-Organization: Inception*, pages 67–114.

Schroecker, Y. and Isbell, C. (2020). Universal value density estimation for imitation learning and goal-conditioned reinforcement learning. *arXiv preprint arXiv:2002.06473*.

Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E., Schaal, S., Levine, S., and Brain, G. (2018). Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1134–1141. IEEE.

Sharma, A., Gu, S., Levine, S., Kumar, V., and Hausman, K. (2019). Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*.

Shumailov, I., Shumaylov, Z., Zhao, Y., Papernot, N., Anderson, R., and Gal, Y. (2024). Ai models collapse when trained on recursively generated data. *Nature*, 631(8022):755–759.

Sikchi, H., Agarwal, S., Jajoo, P., Parajuli, S., Chuck, C., Rudolph, M., Stone, P., Zhang, A., and Niekum, S. (2025). Rlzero: Direct policy inference from language without in-domain supervision.

Sikchi, H., Zheng, Q., Zhang, A., and Niekum, S. (2023). Dual rl: Unification and new methods for reinforcement and imitation learning. *arXiv preprint arXiv:2302.08560*.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. (2020). Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*.

Strehl, A. L. and Littman, M. L. (2008). An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331.

Syed, U., Bowling, M., and Schapire, R. E. (2008). Apprenticeship learning using linear programming. In *Proceedings of the 25th international conference on Machine learning*, pages 1032–1039.

Team, O. M., Ghosh, D., Walke, H., Pertsch, K., Black, K., Mees, O., Dasari, S., Hejna, J., Xu, C., Luo, J., et al. (2023). Octo: An open-source generalist robot policy.

Thrun, S. (2002). Probabilistic robotics. *Communications of the ACM*, 45(3):52–57.

Thrun, S. and Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In *Proceedings of the Fourth Connectionist Models Summer School*, volume 255, page 263. Hillsdale, NJ.

Touati, A. and Ollivier, Y. (2021). Learning one representation to optimize all rewards. *Advances in Neural Information Processing Systems*, 34:13–23.

Touati, A., Rapin, J., and Ollivier, Y. (2023). Does zero-shot reinforcement learning exist? In *The Eleventh International Conference on Learning Representations*.

Venkatraman, A., Hebert, M., and Bagnell, J. (2015). Improving multi-step prediction of learned time series models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

Vincent, P. (2011). A connection between score matching and denoising autoencoders. *Neural Comput.*, 23(7):1661–1674.

Wang, K., Javali, I., Bortkiewicz, M., Eysenbach, B., et al. (2025). 1000 layer networks for self-supervised rl: Scaling depth can enable new goal-reaching capabilities. *arXiv preprint arXiv:2503.14858*.

Wang, Z., Hunt, J. J., and Zhou, M. (2022). Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*.

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

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256.

Wiskott, L. (2022). Slow feature analysis. In *Encyclopedia of Computational Neuroscience*, pages 3142–3143. Springer.

Zhang, M., Vikram, S., Smith, L., Abbeel, P., Johnson, M., and Levine, S. (2019). Solar: Deep structured representations for model-based reinforcement learning. In *International conference on machine learning*, pages 7444–7453. PMLR.

Zheng, C., Eysenbach, B., Walke, H. R., Yin, P., Fang, K., Salakhutdinov, R., and Levine, S. (2024a). Stabilizing contrastive RL: Techniques for robotic goal reaching from offline data. In *The Twelfth International Conference on Learning Representations*.

Zheng, C., Salakhutdinov, R., and Eysenbach, B. (2024b). Contrastive difference predictive coding. In *The Twelfth International Conference on Learning Representations*.

Zheng, C., Tuyls, J., Peng, J., and Eysenbach, B. (2025). Can a MISL fly? analysis and ingredients for mutual information skill learning. In *The Thirteenth International Conference on Learning Representations*.

Zhou, Z., Li, X., and Zare, R. N. (2017). Optimizing chemical reactions with deep reinforcement learning. *ACS central science*, 3(12):1337–1344.

Ziarko, A., Bortkiewicz, M., Zawalski, M., Eysenbach, B., and Miłoś, P. (2025). Contrastive representations for combinatorial reasoning. *In submission*.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., Dey, A. K., et al. (2008). Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA.