Bridging the Gap from Supervised Learning to Control

by

David Brandfonbrener

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September, 2023

Professor Joan Bruna

Acknowledgments

The work behind this thesis would not have been possible without the support of so many people over the last five years. First, I would like to thank my advisor Joan for providing just the right mixture of guidance and freedom to pursue my own interests, and for always having insightful questions that push me to clarify my thinking. I also want to thank all of the members of my committee: Rajesh, Lerrel, Kyunghyun, and Jake. Your advice and collaboration over the last few years has been invaluable.

Next, I've been fortunate to do three internships over my PhD and I'd like to thank each of my internship hosts: Alessandro and Matteo at FAIR, Romain and Remi at MSR, and Jake and Stephen at Google for their mentorship and for broadening my persepectives on research.

Perhaps most directly responsible for the work presented in this thesis are my close collaborators Will, Ofir, Denis, Alberto, and Andrea. Without their contributions, none of the papers in this document would have been possible. I'd also like to single out Will for his exceptional mentorship and friendship as well as collaboration during the middle few years of my PhD.

Next, I'd like to thank all of my colleagues at NYU for creating a great environment for me to grow as a researcher. First, I'd like to thank my cohort of PhD students Mark, Aahlad, Min Jae, Aaron, Mimee, Katrina, and Ethan who have always been so helpful and supportive. Also, I must thank my more senior colleagues Cinjon, Ilya, Elman, Alex, Mukund, Roberta, Jaan, Alfredo, Zhengdao, and Kiante who all set good a precedent for how to do research and offered useful advice whenever

I needed it. And finally I'd also like to thank the junior students who were always willing to discuss research ideas and give me feedback including Mahi, Ben, Raghav, Evgenii, Noah, Ulyana, Anthony, Siddhant, Jyo, Karl, Lei, Lily, and Alex.

I'd also like to thank all my non-NYU affiliated friends for taking my mind off of work sometimes for dinner parties, park hangouts, pickup basketball, or whatever else.

Next, I'd like to thank my family. In particular I'd like to thank my parents Eric and Connie for their tireless support through the ups and downs of my entire education all the way through the PhD. And also my siblings Paul and Clare for always being there for me, I can't wait to see where you both end up once you also finish grad school! I will always be thankful for the time we spent together together back at home during the pandemic in the middle of my PhD.

And last but not least, the biggest thanks go to Rachel for being my closest friend, supporter, and confidant through my whole PhD, even when New Jersey tried to keep us apart. It's cliché, but it's true: I couldn't have done it without you!

Abstract

The combination of deep learning and internet-scale data with supervised learning has led to impressive progress in recent years. However, the potential of this progress has yet to be realized in the context of control problems beyond games that are easy to simulate. This thesis attempts to bridge this gap so as to leverage tools from supervised learning to solve control problems. To do this, we focus on the offline reinforcement learning setting which attempts to learn a control policy from a fixed dataset rather than requiring the policy to learn and collect data at the same time. This removes issues of non-stationary training data and exploration from the control problem, which allows the more straightforward application of tools from supervised learning.

We study this intersection between supervised learning and control from several angles. In the first part of the thesis, we present work on policy learning, focusing on simplified algorithms that look more like standard supervised algorithms. In the second part, we move one step earlier in the pipeline and consider how to best collect datasets for offline reinforcement learning. And in the last part, we consider how to design pretraining objectives to learn representations for downstream offline policy learning. Taken together, these contributions present a nuanced view of the promise and challenges that face the application of machine learning to control problems.

Contents

List of Figures

List of Tables

List of Appendices

1 Introduction

Supervised learning is the dominant paradigm in modern machine learning as most major breakthroughs using deep learning in the last ten years have come from using supervised learning (except for playing board games [Silver et al. 2017]). For example, breakthrough results in image recognition [Krizhevsky et al. 2012], language modeling [Brown et al. 2020], and protein structure prediction [Jumper et al. 2021] all rely on supervised learning as the core algorithmic technique.

However, not all problems where we may want to apply machine learning can be neatly cast as supervised learning problems. In particular, control problems, where an algorithm must interact with the environment by making decisions that affect the state of the world, necessitate going beyond supervised learning for several reasons. First, in a standard control problem we are not handed a dataset sampled from the data generating distribution, but rather must interact with the environment to collect our own data. Second, control problems often require temporal credit assignment (usually accomplished via dynamic programming) to determine which action in a sequence of actions was responsible for an outcome. In contrast, supervised learning problems usually come with a dataset sampled i.i.d. from the target distribution and rely on predicting one observed target variable without needing to worry about credit assignment.

In this thesis, we will instead focus on a middle ground between supervised learning and control: the offline reinforcement learning problem. This setting allows us to approach control problems from fixed datasets where tools from supervised learning are more directly applicable. Of course, this simplification of the control problem comes with a loss of generality, and we will examine when exactly this approach makes sense in practice. Before going into the main contributions in more detail, this introductory section will present some background and motivation for the work presented in this thesis.

1.1 Defining terms: the supervised and control settings

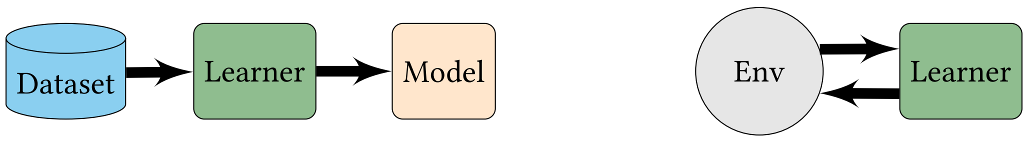


Figure 1.1: Diagrams of the supervised learning (left) and control (right) pipelines. In the supervised pipeline, a fixed dataset is ingested by the learner to produce a model. In the control pipeline, the learner interacts online with an environment ("Env") to produce a policy.

First, we need to define the two settings that we hope to bridge in this thesis. It is instructive to begin at the absolute highest level of abstraction as to how data moves through the learning pipeline in each of the two settings. As seen in Figure 1.1 supervised learning is a paradigm where our learning algorithm ingests a static dataset and produces a model. In contrast, learned control (e.g. reinforcement learning [Sutton and Barto 2018]) is traditionally viewed as an online process where a learner is continually collecting data from an environment and updating its learned model or models.

To be more rigorous, we can more precisely define each of the two settings as follows:

- A supervised learning pipeline is defined by a dataset $\mathcal{D}$ and a learner. The dataset contains labeled pairs of inputs $x \in \mathcal{X}$ and labels $y \in \mathcal{Y}$. The learner contains a parametric model class $\mathcal{F} = \{f_\theta : \theta \in \Theta\}$, a loss function $\ell(\hat{y}, y)$, and an optimization algorithm $\mathcal{A}$ (usually stochastic gradient descent). Learning then proceeds by using the optimization algorithm to minimize the empirical loss over the dataset: $\hat{L}(\theta) := \sum_{(x,y)\in\mathcal{D}} \ell(f_\theta(x), y)$.

- A control pipeline instead involves an environment and a learner. The environment is typically a Markov Decision Process (MDP) where at each timestep the environment yields a state $s \in \mathcal{S}$ to the learner, which returns an action $a \in \mathcal{A}$ and then the environment subsequently returns the next state $s' \in \mathcal{S}$ according to some dynamics T so that $s' \sim T(\cdot|s,a)$ and a reward $r(s,a)$. The goal is to maximize cumulative reward over some horizon H. The learner selects actions according to some parametric policy π_θ, and in doing so builds up a dataset $\mathcal{D}$ of historical experience of transitions (s,a,r,s') (this dataset is also known as a replay buffer). Simultaneously, the learner is updating parametric policy π_θ so as to maximize reward. The selected actions must balance the need to "explore" new states and actions while learning how to "exploit" the maximum expected return. We will leave precise details of the algorithm vague for now since, unlike supervised learning, there are very many options for the internal workings of the learner (e.g. model-based vs. model-free, policy-gradient vs. actor-critic, etc.).

Note: each chapter may consider slight variations of these and related settings, but we will be explicit about the specific setting and notation considered within each chapter.

There is a clear tension between these two high level learning paradigms. Most importantly, the control setting emphasizes the closed loop feedback between the learning algorithm and the environment while the supervised setting cleanly separates the data collection process from the learning process. This fundamental difference yields substantial differences in the methods that are traditionally used in the two settings. Resolving these tensions will be key to accomplishing the main goal of this thesis to understand how to leverage tools from supervised learning to solve control problems.

1.2 Key components of success in supervised learning

So as to leverage the successes of supervised learning, it is important to have a high level understanding of which components of the standard supervised learning pipeline described above are crucial to success, with a special focus on those components that are difficult to achieve in control problems. We argue that the key components of a successful recipe for a supervised learning pipeline are good datasets, simple learning algorithms, and pretraining. We now discuss each in turn.

Datasets. One of the most fundamental components of the success of supervised learning in the last ten years or so has been the creation of better datasets in terms of both quality and quantity. This perhaps began with ImageNet for image classification [Deng et al. 2009] and has steadily improved since then. For example, datasets used in the pipeline of training image classifiers now can have billions of images [Sun et al. 2017; Yalniz et al. 2019]. Advances in natural language processing, and particularly language modeling, have also been facilitated by internet-scale datasets [Raffel et al. 2019; Radford et al. 2019; Brown et al. 2020]. And in other application areas like protein structure prediction where data cannot be scraped from the internet, a consortium of scientists collecting data together to build large datasets has been key to recent progress [wwPDB consortium 2018]. Simply put, large and high quality datasets are a prerequisite and key ingredient to all of the recent progress in applying supervised learning.

Simple learning algorithms. On top of the aforementioned datasets, supervised learning relies on simple learning algorithms that are easily scaled up. This begins with simple model architecture building blocks like convolutional neural nets [LeCun et al. 2015] or transformers [Vaswani et al. 2017] which facilitate the creation of large models. Then the standard recipe for supervised learning simply runs stochastic gradient descent or moderately more complicated variants like Adam [Kingma and Ba 2014] to train the models. This simple recipe is easy to scale up as larger datasets and compute resources become available, an intuition which has recently been formalized into scaling laws for neural net training [Kaplan et al. 2020]. The simplicity and stability of the overall supervised learning pipeline has facilitated the impressive scaling up that has led to recent successes.

Pretraining. One component of the modern supervised learning pipeline that we have yet to discuss is pretraining. Rather than directly training on the target task (where data may be scarce), it is often useful to pretrain a supervised (or self-supervised) model on some pretraining task on much larger datasets. This has been incredibly effective in NLP using variants of language modeling as the supervised pretraining objective [Devlin et al. 2018; Raffel et al. 2019] and in computer vision with pretraining objectives like contrastive loss or masked reconstruction [Chen et al. 2020a; He et al. 2022]. The ability to easily transfer pretrained features from large pretraining datasets to target tasks where data may be more scarce has also been important for widespread application of advances in supervised learning.

These three components of datasets, algorithms, and pretraining give a high level explanation of much of the success in supervised learning. But, in this thesis we want to instead consider how to solve control problems. The question then becomes whether we can set up control problems in such a way as to gain access to high quality datasets, simple algorithms, and pretraining.

1.3 Challenges for applying learning to control

As explained earlier, control problems require dealing with closed loop feedback between the learner and the environment. This yields a host of issues when attempting to apply tools from supervised learning since these tools do not work well out of the box in systems with feedback. Specifically, and in direct contrast to the components of success in supervised learning discussed above, this can cause issues due to non-stationary data, the need for exploration, the need to perform temporal credit assignment, and challenges to the pretraining-finetuning paradigm. We now discuss each in turn.

Non-stationary data. In a standard reinforcement learning setup, the learner interacts with the environment while simultaneously updating the learned policy. This leads to a constantly shifting distribution of data, since the distribution of data is induced by the learned policy. This is a major departure from supervised learning where the dataset is carefully curated by human data collectors and then held fixed during learning. As a result it is not possible to have the same sort of curated data as an input to the learner in the standard control setup. This is a major issue since one of the most effective ways of guiding what a model will learn is being able to curate what goes into the training data. Moreover, since the data that is being collected online in a non-stationary way, it is difficult to apply standard supervised learning algorithms that often struggle in the face of a shifting distribution [Quinonero-Candela et al. 2008].

Exploration. Another issue that arises due to the simultaneous collection of data and learning is the need to explore the environment. Unlike supervised learning where the data distribution of interest is given to the learner by the dataset, the learner in a control problem must essentially discover the optimal distribution over trajectories (out of all possible distributions) in the course of learning the policy. This problem is substantially harder since it may require being able to visit every possible state of the world in the worst case.

Temporal credit assignment. Adding to the complexity of the control setting is the issue of credit assignment. The learner must reason about how a sequence of actions affects the state of the environment and which actions are most responsible for the outcomes of interest. The most common approach to perform this reasoning is to use variants of dynamic programming where the model estimates the value of each state-action pair (usually denoted $Q(s, a)$) by iteratively propagating estimated values backwards in time over observed transitions (i.e. moving $Q(s, a)$ towards $r + Q(s', a')$) [Sutton 1988]. This yields learning algorithms that look far different from simply running gradient descent to minimize a loss. Dynamic programming algorithms are instead seeking a fixed point of the Bellman equation and rely on repeatedly constructing regression problems where the targets themselves are defined by the model, which can cause instability [van Hasselt et al. 2018; Achiam et al. 2019; Brandfonbrener and Bruna 2020]. Moreover, this often results in algorithms with multiple neural models (e.g. actor, critic, and target network [Haarnoja et al. 2018]) all trained simultaneously with complicated dependence on each other. These substantially more complicated algorithms that are needed to perform credit assignment have proven more difficult to tune and scale than the standard gradient descent recipe used in supervised learning.

Pretraining. While pretraining has been an important component of supervised learning pipelines, it has been less clear how to incorporate pretraining for control problems. There have been many proposed methods, e.g. Stooke et al. [2020]; Liu and Abbeel [2021a]; Yarats et al. [2021a]; Agarwal et al. [2022]. However, there have yet to be similar breakthroughs as in supervised learning. Much of this is likely downstream of the two issues listed above, namely that it is unclear how to generate and use large, high quality datasets in control problems and that dealing with temporal credit assignment adds a layer of algorithmic complexity. The lack of a standard way to apply pretraining methods, or to transfer pretrained policies and representations to new tasks has made it difficult to apply learning more broadly to new control problems.

Comparing control and supervised learning, we see that there are substantial roadblocks towards applying tools from supervised learning in control problems, even as subroutines. Instead of nicely curated datasets we have non-stationary data that is collected by the learner, shifts as the policy is updated, and requires exploration. Instead of simple gradient descent we have more complicated dynamic programming algorithms. Instead of well established pretraining techniques we have a much less well-explored space of pretraining possibilities. To begin to resolve these differences, we will instead change the setting to a simplified version of the control problem that removes some of these issues.

1.4 Offline reinforcement learning: a middleground

So far we have seen that control problems yield several undesirable characteristics for applying techniques from supervised learning. To begin to resolve this divide, we will introduce a third setting which is a sort of middleground between supervised learning and control, namely offline reinforcement learning. Offline RL [Levine et al. 2020], as illustrated in Figure 1.2, splits a control problem into two phases. First, there is a data collection phase where some algorithm, human, or system collects a dataset by interacting with an environment. Then, there is a policy learning phase where a learner ingests this dataset of interactions and learns a policy that could then be deployed in the environment.

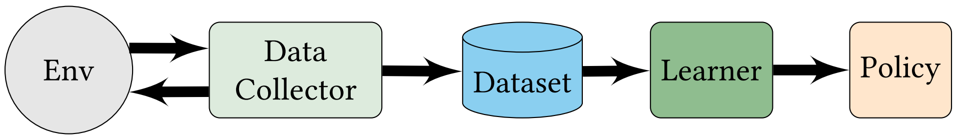


Figure 1.2: A schematic diagram of the offline RL pipeline.

The offline RL setting has several desirable qualities. By explicitly breaking the dependence between data collection and policy learning, we can resolve the non-stationarity issues that arise in the online control setting and separate exploration from learning. But, the setting still maintains the idea from control of using interaction data to perform temporal credit assignment to learn policies that can interact with a system in a closed loop in a meaningful way that goes beyond supervised learning. In this way, offline RL provides an interesting testbed to attempt to use supervised learning methods as subroutines to solve control problems.

As presented here, this setting still does not include any separate pretraining and finetuning phases, which have often been beneficial to practical application of supervised learning techniques. Figure 1.3 illustrates a larger training pipeline that incorporates separate pretraining and finetuning phases for offline RL. We will also study this more general setting to get a better idea of how to relate pretraining to finetuning when the data is structured as MDP transitions rather than labeled datapoints and when the downstream finetuning task is to solve a control problem rather than classification or regression.

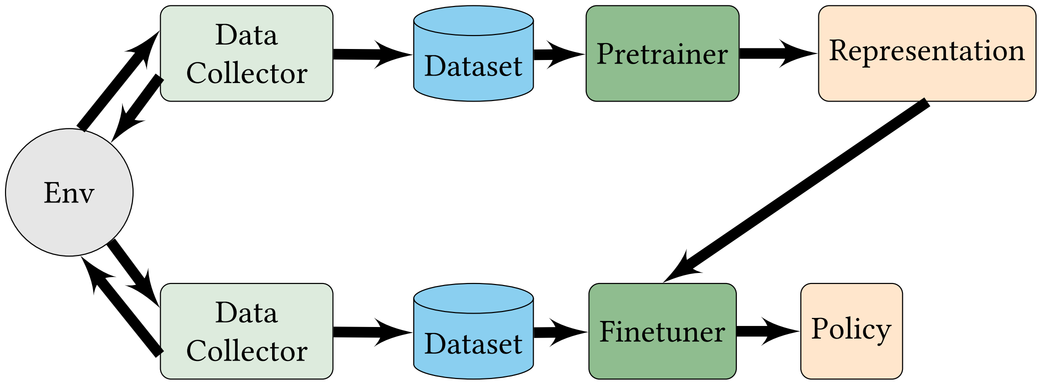


Figure 1.3: A schematic diagram of an offline RL pipeline with pretraining and finetuning phases.

1.5 Summary of contributions

Now that the setting and motivation are clear, we can preview the main contributions of this thesis. We break the results into three parts: policy learning, data collection, and pretraining. These can be mapped onto the learner component of Figure 1.2, the data collector component of Figure 1.2, and the pretrainer component of Figure 1.3 respectively.

Policy learning. First we consider how ideas from supervised learning can be relevant to control via offline RL in the policy learning phase. In this section we assume access to a fixed dataset of experience and try to learn a policy. The first chapter in this part presents work from

Brandfonbrener et al. [2021] which illustrates how the standard approach of running iterative RL algorithms that were designed for online learning in the offline setting can often fail. Instead, a simpler baseline of training for only one step of policy iteration (so that the policy improvement step looks much more like standard supervised learning) can often be as effective if not more so, while being substantially simpler. The second chapter presents work from Brandfonbrener et al. [2022a] which considers whether offline RL problem can be solved directly with return-conditioned supervised learning. We find that while this supervised approach is sometimes sufficient, it often fails to perform adequate credit assignment.

Data collection. Next we examine data collection for offline RL. Data collection must resolve a tension between the desire for coverage of all possible behaviors for generality and demonstrations of expert behaviors for efficiency. We consider two extreme settings: trying to collect fully exploratory data for maximum coverage and trying to leverage an expert demonstrator with maximum efficiency. The first chapter of this part presents Yarats et al. [2022] which studies exploratory data collection. While we find that exploratory data facilitates multitask learning with simple policy learning algorithms, it is difficult to scale this technique to complicated environments. The second chapter presents Brandfonbrener et al. [2022b] which attempts to leverage an expert demonstrator as efficiently as possible to solve a single task. We find it is important to collect failures that are as similar as possible to successes so as to isolate the decision boundary and present a method for actualizing this in a specific robotic manipulation task.

Pretraining. Finally, we step back to the larger pretraining-finetuning pipeline presented in Figure 1.3. We present results from Brandfonbrener et al. [2023] which compare a variety of pretraining objectives for representation learning for downstream control. For simplicity we consider imitation learning rather than the more general reinforcement learning problem, but we emphasize learning from diverse multitask pretraining data. Through empirical and theoretical analysis, we find that modeling the inverse dynamics of a system (predicting the action given observations from before and after the action was taken) makes for a good pretraining objective for representation learning.

Cumulatively, these results paint a picture of how to begin to bring tools from supervised learning into control settings in well-motivated ways. Of course the results are often nuanced and which method works best depends on the precise nature of each problem and dataset. But, taken as a whole, the contributions indicate a variety of principles that can be incorporated into designing learning systems for control problems by being cognizant of the strengths and limitations of the tools from deep supervised learning.

Part I

Policy learning

2 | Offline reinforcement learning without off-policy evaluation

2.1 Introduction

An important step towards effective real-world RL is to improve sample efficiency. One avenue towards this goal is offline RL (also known as batch RL) where we attempt to learn a new policy from data collected by some other behavior policy without interacting with the environment. Recent work in offline RL is well summarized by [Levine et al. 2020].

In this paper, we challenge the dominant paradigm in the deep offline RL literature that primarily relies on actor-critic style algorithms that alternate between policy evaluation and policy improvement [Fujimoto et al. 2018a, 2019a; Peng et al. 2019; Kumar et al. 2019a, 2020; Wang et al. 2020c; Wu et al. 2019; Kostrikov et al. 2021b; Jaques et al. 2019; Siegel et al. 2020; Nachum et al. 2019]. All these algorithms rely heavily on off-policy evaluation to learn the critic. Instead, we find that a simple baseline which only performs one step of policy improvement using the behavior Q function often outperforms the more complicated iterative algorithms. Explicitly, we find that our one-step algorithm beats prior results of iterative algorithms on most of the gym-mujoco [Brockman et al. 2016] and Adroit [Rajeswaran et al. 2017] tasks in the the D4RL benchmark suite [Fu et al. 2020].

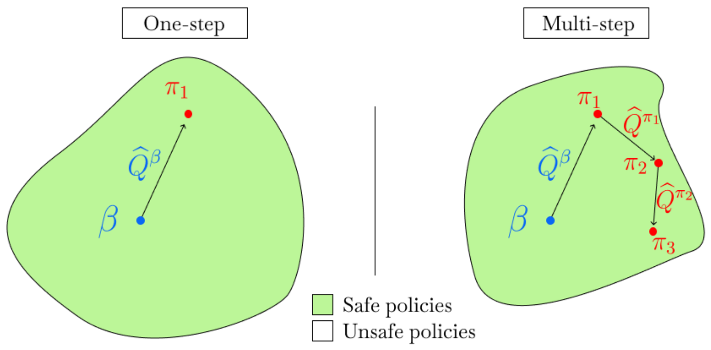


Figure 2.1: A cartoon illustration of the difference between one-step and multi-step methods. All algorithms constrain themselves to a neighborhood of "safe" policies around β. A one-step approach (left) only uses the on-policy $\widehat{Q}^{\beta}$, while a multi-step approach (right) repeatedly uses off-policy $\widehat{Q}^{\pi_i}$.

We then dive deeper to understand why such a simple baseline is effective. First, we examine what goes wrong for the iterative algorithms. When these algorithms struggle, it is often due to poor off-policy evaluation leading to inaccurate Q values. We attribute this to two causes: (1) distribution shift between the behavior policy and the policy to be evaluated, and (2) iterative error exploitation whereby policy optimization introduces bias and dynamic programming propagates this bias across the state space. We show that empirically both issues exist in the benchmark tasks and that one way to avoid these issues is to simply avoid off-policy evaluation entirely.

Finally, we recognize that while the the one-step algorithm is a strong baseline, it is not always the best choice. In the final section we provide some guidance about when iterative algorithms can perform better than the simple one-step baseline. Namely, when the dataset is large and behavior policy has good coverage of the state-action space, then off-policy evaluation can succeed and iterative algorithms can be effective. In contrast, if the behavior policy is already fairly good, but as a result does not have full coverage, then one-step algorithms are often preferable.

Our main contributions are:

- A demonstration that a simple baseline of one step of policy improvement outperforms more complicated iterative algorithms on a broad set of offline RL problems.

- An examination of failure modes of off-policy evaluation in iterative offline RL algorithms.
- A description of when one-step algorithms are likely to outperform iterative approaches.

2.2 Setting and notation

We will consider an offline RL setup as follows. Let $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \rho, P, R, \gamma\}$ be a discounted infinite-horizon MDP. In this work we focus on applications in continuous control, so we will generally assume that both $\mathcal{S}$ and $\mathcal{A}$ are continuous and bounded. We consider the offline setting where rather than interacting with $\mathcal{M}$, we only have access to a dataset D_N of N tuples of (s_i, a_i, r_i) collected by some behavior policy β with initial state distribution ρ. Let $r(s,a) = \mathbb{E}_{r|s,a}[r]$ be the expected reward. Define the state-action value function for any policy π by $Q^\pi(s,a) := \mathbb{E}_{P,\pi|s_0=s,\ a_0=a}[\sum_{t=0}^\infty \gamma^t r(s_t,a_t)]$. The objective is to maximize the expected return J of the learned policy:

$$J(\pi) := \mathop{\mathbb{E}}_{\rho,P,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right] = \mathop{\mathbb{E}}_{\substack{s\sim\rho \\ a\sim\pi|s}} [Q^\pi(s,a)]. \tag{2.1}$$

Following [Fu et al. 2020] and others in this line of work, we allow access to the environment to tune a small (< 10) set of hyperparameters. See [Paine et al. 2020] for a discussion of the active area of research on hyperparameter tuning for offline RL. We also discuss this further in Appendix A.3.

2.3 Related work

Iterative algorithms. Most prior work on deep offline RL consists of iterative actor-critic algorithms. The primary innovation of each paper is to propose a different mechanism to ensure that the learned policy does not stray too far from the data generated by the behavior policy. Broadly, we group these methods into three camps: policy constraints/regularization, modifications of imitation learning, and Q regularization:

1. The majority of prior work acts directly on the policy. Some authors have proposed explicit constraints on the learned policy to only select actions where (s, a) has sufficient support under the data generating distribution [Fujimoto et al. 2018a, 2019a; Laroche et al. 2019]. Another proposal is to regularize the learned policy towards the behavior policy [Wu et al. 2019] usually either with a KL divergence [Jaques et al. 2019] or MMD [Kumar et al. 2019a]. This is a very straighforward way to stay close to the behavior with a hyperparameter that determines just how close. All of these algorithms are iterative and rely on off-policy evaluation.

2. [Siegel et al. 2020; Wang et al. 2020c; Chen et al. 2020b] all use algorithms that filter out datapoints with low Q values and then perform imitation learning. [Wang et al. 2018; Peng et al. 2019] use a weighted imitation learning algorithm where the weights are determined by exponentiated Q values. These algorithms are iterative.

3. Another way to prevent the learned policy from choosing unknown actions is to incorporate some form of regularization to encourage staying near the behavior and being pessimistic about unknown state, action pairs [Wu et al. 2019; Nachum et al. 2019; Kumar et al. 2020; Kostrikov et al. 2021b; Gulcehre et al. 2021]. However, being able to properly quantify uncertainty about unknown states is notoriously difficult when dealing with neural network value functions [Buckman et al. 2020].

One-step algorithms. Some recent work has also noted that optimizing policies based on the behavior value function can perform surprisingly well. As we do, [Goo and Niekum 2020] studies the continuous control tasks from the D4RL benchmark, but they examine a complicated algorithm involving ensembles, distributional Q functions, and a novel regularization technique.

In contrast, we analyze a substantially simpler algorithm and get better performance on the D4RL tasks. We also focus more of our contribution on understanding and explaining this performance. [Gulcehre et al. 2021] studies the discrete action setting and finds that a one-step algorithm (which they call "behavior value estimation") outperforms prior work on Atari games and other discrete action tasks from the RL Unplugged benchmark [Gulcehre et al. 2020]. They also introduce a novel regularizer for the evaluation step. In contrast, we consider the continuous control setting. This is a substantial difference in setting since continuous control requires actor-critic algorithms with parametric policies while in the discrete setting the policy improvement step can be computed exactly from the Q function. Moreover, while [Gulcehre et al. 2021] attribute the poor performance of iterative algorithms to "overestimation", we define and separate the issues of distribution shift and iterative error exploitation which can combine to cause overestimation. This separation helps to expose the difference between the fundamental limits of off-policy evaluation from the specific problems induced by iterative algorithms, and will hopefully be a useful distinction to inspire future work. Finally, a one-step variant is also briefly discussed in [Nadjahi et al. 2019], but is not the focus of that work.

There are also important connections between the one-step algorithm and the literature on conservative policy improvement [Kakade and Langford 2002; Schulman et al. 2015; Achiam et al. 2017], which we discuss in more detail in Appendix A.2.

2.4 Defining the algorithms

In this section we provide a unified algorithmic template for model-free offline RL algorithms as offline approximate modified policy iteration. We show how this template captures our one-step algorithm as well as a multi-step policy iteration algorithm and an iterative actor-critic algorithm. Then any choice of policy evaluation and policy improvement operators can be used to define one-step, multi-step, and iterative algorithms.

Algorithm 1: OAMPI

```
input: $K$, dataset $D_N$, estimated behavior $\hat{\beta}$
Set $\pi_0 = \hat{\beta}$. Initialize $\widehat{Q}^{\pi_{-1}}$ randomly.
for $k = 1, \dots, K$ do
    Policy evaluation: $\widehat{Q}^{\pi_{k-1}} = \mathcal{E}(\pi_{k-1}, D_N, \widehat{Q}^{\pi_{k-2}})$
    Policy improvement: $\pi_k = \mathcal{I}(\widehat{Q}^{\pi_{k-1}}, \hat{\beta}, D_N, \pi_{k-1})$
end
```

2.4.1 Algorithmic template

We consider a generic offline approximate modified policy iteration (OAMPI) scheme, shown in Algorithm 1 (and based off of [Puterman and Shin 1978; Scherrer et al. 2012]). Essentially the algorithm alternates between two steps. First, there is a policy evaluation step where we estimate the Q function of the current policy π_{k-1} by $\widehat{Q}^{\pi_{k-1}}$ using only the dataset D_N. Implementations also often use the prior Q estimate $\widehat{Q}^{\pi_{k-2}}$ to warm-start the approximation process. Second, there is a policy improvement step. This step takes in the estimated Q function $\widehat{Q}^{\pi_{k-1}}$, the estimated behavior $\hat{\beta}$, and the dataset D_N and produces a new policy π_k. Again an algorithm may use π_{k-1} to warm-start the optimization. Moreover, we expect this improvement step to be regularized or constrained to ensure that π_k remains in the support of β and D_N. Choices for this step are discussed below. Now we discuss a few ways to instantiate the template.

One-step. The simplest algorithm sets the number of iterations $K = 1$. We learn $\hat{\beta}$ by maximum likelihood and train the policy evaluation step to estimate Q^{β}. Then we use any one of the policy improvement operators discussed below to learn π_1. Importantly, this algorithm completely avoids off-policy evaluation.

Multi-step. The multi-step algorithm now sets $K > 1$. The evaluation operator must evaluate off-policy since D_N is collected by β, but evaluation steps for $K \geq 2$ require evaluating policies $\pi_{k-1} \neq \beta$. Each iteration is trained to convergence in both the estimation and improvement steps.

Iterative actor-critic. An actor critic approach looks somewhat like the multi-step algorithm, but does not attempt to train to convergence at each iteration and uses a much larger K. Here each iteration consists of one gradient step to update the Q estimate and one gradient step to improve the policy. Since all of the evaluation and improvement operators that we consider are gradient-based, this algorithm can adapt the same evaluation and improvement operators used by the multi-step algorithm. Most algorithms from the literature fall into this category [Fujimoto et al. 2018a; Kumar et al. 2019a, 2020; Wu et al. 2019; Wang et al. 2020c; Siegel et al. 2020].

2.4.2 Policy evaluation operator

Following prior work on continuous state and action problems, we always evaluate by simple fitted Q evaluation [Fujimoto et al. 2018a; Kumar et al. 2019a; Siegel et al. 2020; Wang et al. 2020c; Paine et al. 2020; Wang et al. 2021]. In practice this is optimized by TD-style learning with the use of a target network [Mnih et al. 2015] as in DDPG [Lillicrap et al. 2015]. We do not use any double Q learning or Q ensembles [Fujimoto et al. 2018b]. For the one-step and multi-step algorithms we train the evaluation procedure to convergence on each iteration and for the iterative algorithm each iteration takes a single stochastic gradient step. See [Voloshin et al. 2019; Wang et al. 2021] for more comprehensive examinations of policy evaluation and some evidence that this simple fitted Q iteration approach is reasonable. It is an interesting direction for future work to consider other operators that use things like importance weighting [Munos et al. 2016] or pessimism [Kumar et al. 2020; Buckman et al. 2020].

2.4.3 Policy improvement operators

To instantiate the template, we also need to choose a specific policy improvement operator $\mathcal{I}$. We consider the following improvement operators selected from those discussed in the related work section. Each operator has a hyperparameter controlling deviation from the behavior policy.

Behavior cloning. The simplest baseline worth including is to just return $\hat{\beta}$ as the new policy π. Any policy improvement operator ought to perform at least as well as this baseline.

Constrained policy updates. Algorithms like BCQ [Fujimoto et al. 2018a] and SPIBB [Laroche et al. 2019] constrain the policy updates to be within the support of the data/behavior. In favor of simplicity, we implement a simplified version of the BCQ algorithm that removes the "perturbation network" which we call Easy BCQ. We define a new policy $\hat{\pi}_k^M$ by drawing M samples from $\hat{\beta}$ and then executing the one with the highest value according to $\widehat{Q}^{\beta}$. Explicitly:

$$\hat{\pi}_k^M(a|s) = \mathbb{1}[a = \arg\max_{a_j}\{\widehat{Q}^{\pi_{k-1}}(s, a_j) : a_j \sim \pi_{k-1}(\cdot|s),\ 1 \leq j \leq M\}]. \tag{2.2}$$

Regularized policy updates. Another common idea proposed in the literature is to regularize towards the behavior policy [Wu et al. 2019; Jaques et al. 2019; Kumar et al. 2019a]. For a general divergence D we can define an algorithm that maximizes a regularized objective:

$$\hat{\pi}_k^\alpha = \arg\max_\pi \sum_i \mathop{\mathbb{E}}_{a\sim\pi|s}[\widehat{Q}^{\pi_{k-1}}(s_i, a)] - \alpha D(\hat{\beta}(\cdot|s_i), \pi(\cdot|s_i)) \tag{2.3}$$

A comprehensive review of different variants of this method can be found in [Wu et al. 2019] which does not find dramatic differences across regularization techniques. In practice, we will use reverse KL divergence, i.e. $KL(\pi(\cdot|s_i)\|\hat{\beta}(\cdot|s_i))$. To compute the reverse KL, we draw samples from $\pi(\cdot|s_i)$ and use the density estimate $\hat{\beta}$ to compute the divergence. Intuitively, this regularization forces π to remain within the support of β rather than incentivizing π to cover β.

Variants of imitation learning. Another idea, proposed by [Wang et al. 2018; Siegel et al. 2020; Wang et al. 2020c; Chen et al. 2020b] is to modify an imitation learning algorithm either by filtering or weighting the observed actions to incentivize policy improvement. The weighted version that we implement uses exponentiated advantage estimates to weight the observed actions:

$$\hat{\pi}_k^\tau = \arg\max_\pi \sum_i \exp(\tau(\widehat{Q}^{\pi_{k-1}}(s_i, a_i) - \widehat{V}(s_i))) \log \pi(a_i|s_i). \tag{2.4}$$

With these definitions, we can now move on to testing various combinations of algorithmic template (one-step, multi-step, or iterative) and improvement operator (Easy BCQ, reverse KL regularization, or exponentially weighted imitation).

2.5 Benchmark Results

Our main empirical finding is that one step of policy improvement is sufficient to beat state of the art results on much of the D4RL benchmark suite [Fu et al. 2020]. This is striking since prior work focuses on iteratively estimating the Q function of the current policy iterate, but we only use one step derived from $\widehat{Q}^\beta$. Results are shown in Table 2.1. Full experimental details are in Appendix A.3 and code can be found at `https://github.com/davidbrandfonbrener/onestep-rl`.

As we can see in the table, all of the one-step algorithms usually outperform the best iterative algorithms tested by [Fu et al. 2020]. The one notable exception is the case of random data (especially on halfcheetah), where iterative algorithms have a clear advantage. We will discuss potential causes of this further in Section 2.7.

To give a more direct comparison that controls for any potential implementation details, we use our implementation of reverse KL regularization to create multi-step and iterative algorithms. We are not using algorithmic modifications like Q ensembles, regularized Q values, or early stopping that have been used in prior work. But, our iterative algorithm recovers similar performance to prior regularized actor-critic approaches. These results are shown in Table 2.2.

Put together, these results immediately suggest some guidance to the practitioner: it is worthwhile

Table 2.1: Results of one-step algorithms on the D4RL benchmark. The first column gives the best results across several iterative algorithms considered in [Fu et al. 2020]. Each algorithm is tuned over 6 values of their respective hyperparameter. We report the mean and standard error over 10 seeds of the training process and using 100 evaluation episodes per seed. We **bold** the best result on each dataset and blue any result where a one-step algorithm beat the best reported iterative result from [Fu et al. 2020]. We use m for medium, m-e for medium-expert, m-re for medium-replay, r for random, and c for cloned.

	Iterative	One-step			
	[Fu et al. 2020]	BC	Easy BCQ	Rev. KL Reg	Exp. Weight
halfcheetah-m	46.3	42.1 ± 0.1	52.6 ± 0.1	**55.6 ± 0.2**	48.6 ± 0.0
walker2d-m	81.1	70.2 ± 1.3	**86.9 ± 0.4**	85.6 ± 0.4	80.3 ± 1.1
hopper-m	58.8	49.8 ± 0.6	69.7 ± 2.1	**83.3 ± 1.4**	56.7 ± 0.8
halfcheetah-m-e	64.7	60.1 ± 0.8	77.0 ± 0.9	**93.5 ± 0.1**	91.7 ± 0.9
walker2d-m-e	111.0	93.6 ± 5.6	111.8 ± 0.2	110.9 ± 0.1	**112.9 ± 0.2**
hopper-m-e	**111.9**	48.1 ± 1.5	81.4 ± 1.9	102.1 ± 1.3	83.1 ± 7.0
halfcheetah-m-re	**47.7**	34.9 ± 0.3	38.4 ± 0.3	42.4 ± 0.1	38.6 ± 0.5
walker2d-m-re	26.7	23.9 ± 1.6	66.4 ± 2.0	**71.6 ± 3.1**	49.3 ± 3.5
hopper-m-re	48.6	21.2 ± 1.3	77.3 ± 2.7	71.0 ± 8.1	**94.1 ± 2.4**
halfcheetah-r	**35.4**	2.2 ± 0.0	5.4 ± 0.1	6.9 ± 1.0	3.7 ± 0.2
walker2d-r	**7.3**	0.7 ± 0.1	4.2 ± 0.2	6.1 ± 0.3	5.2 ± 0.2
hopper-r	**12.2**	2.6 ± 0.4	6.7 ± 0.1	7.8 ± 0.3	5.6 ± 0.6
pen-c	56.9	49.3 ± 2.2	**67.0 ± 1.1**	55.3 ± 1.9	54.7 ± 2.3
hammer-c	2.1	0.5 ± 0.1	**2.8 ± 0.5**	0.2 ± 0.0	1.2 ± 0.2
relocate-c	-0.1	0.0 ± 0.0	**0.3 ± 0.0**	0.1 ± 0.0	0.1 ± 0.0
door-c	**0.4**	0.0 ± 0.0	**0.4 ± 0.2**	0.0 ± 0.1	0.1 ± 0.1

to run the one-step algorithm as a baseline before trying something more elaborate. The one-step algorithm is substantially simpler than prior work, but frequently achieves better performance.

2.6 What goes wrong for iterative algorithms?

The benchmark experiments show that one step of policy improvement often beats iterative and multi-step algorithms. In this section we dive deeper to understand why this happens. First, by examining the learning curves of each of the algorithms we note that iterative algorithms require stronger regularization to avoid instability. Then we identify two causes of this instability:

Table 2.2: Results of reverse KL regularization on the D4RL benchmark across one-step, multi-step, and iterative algorithms. Again we run 6 hyperparameters and report the mean and standard error across 10 seeds using 100 evaluation episodes.

	One-step	Multi-step	Iterative
halfcheetah-m	**55.6** ± **0.2**	40.8 ± 8.6	47.4 ± 3.5
walker2d-m	**85.6** ± **0.4**	75.9 ± 0.5	75.4 ± 0.8
hopper-m	**83.3** ± **1.4**	53.0 ± 1.0	54.2 ± 0.6
halfcheetah-m-e	**93.5** ± **0.1**	**93.6** ± **0.3**	**93.6** ± **0.2**
walker2d-m-e	**110.9** ± **0.1**	76.3 ± 15.9	108.2 ± 0.3
hopper-m-e	**102.1** ± **1.3**	**101.3** ± **3.9**	82.7 ± 7.4
halfcheetah-r	6.9 ± 1.0	13.7 ± 1.7	**16.3** ± **1.6**
walker2d-r	**6.1** ± **0.3**	5.0 ± 0.3	5.1 ± 0.3
hopper-r	7.8 ± 0.3	**15.4** ± **2.9**	9.7 ± 0.1

distribution shift and *iterative error exploitation.*

Distribution shift causes evaluation error by reducing the effective sample size in the fixed dataset for evaluating the current policy and has been extensively considered in prior work as discussed below. Iterative error exploitation occurs when we repeatedly optimize policies against our Q estimates and exploit their errors. This introduces a bias towards overestimation at each step (much like the training error in supervised learning is biased to be lower than the test error). Moreover, by iteratively re-using the data and using prior Q estimates to warmstart training at each step, the errors from one step are amplified at the next. This type of error is particular to multi-step and iterative algorithms.

2.6.1 Learning curves and hyperparameter sensitivity

To begin to understand why iterative and multi-step algorithms can fail it is instructive to look at the learning curves. As shown in Figure 2.2, we often observe that the iterative algorithm will begin to learn and then crash. Regularization can help to prevent this crash since strong enough regularization towards the behavior policy ensures that the evaluation is nearly on-policy.

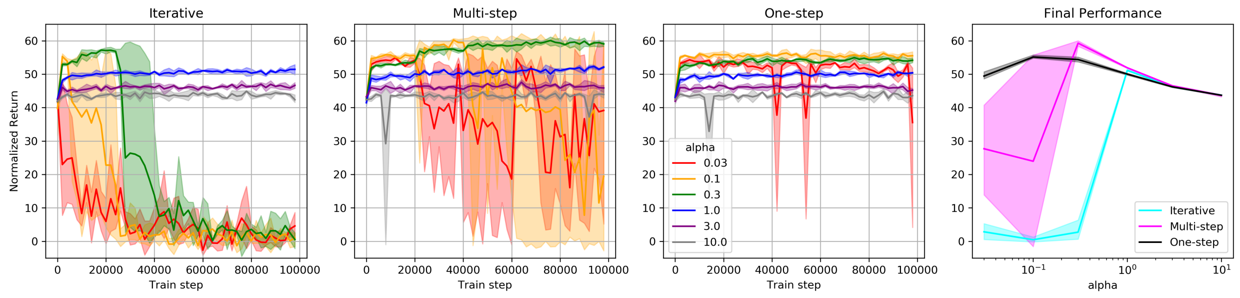


Figure 2.2: Learning curves and final performance on halfcheetah-medium across different algorithms and regularization hyperparameters (all using the reverse KL regularized improvement operator). Error bars show min and max over 3 seeds. Similar figures for other datasets from D4RL can be found in Appendix A.4.

In contrast, the one-step algorithm is more robust to the regularization hyperparameter. The rightmost panel of the figure shows this clearly. While iterative and multi-step algorithms can have their performance degrade very rapidly with the wrong setting of the hyperparameter, the one-step approach is more stable. Moreover, we usually find that the optimal setting of the regularization hyperparameter is lower for the one-step algorithm than the iterative or multi-step approaches.

2.6.2 Distribution shift

Any algorithm that relies on off-policy evaluation will struggle with distribution shift in the evaluation step. Trying to evaluate a policy that is substantially different from the behavior reduces the effective sample size and increases the variance of the estimates. Explicitly, by distribution shift we mean the shift between the behavior distribution (the distribution over state-action pairs in the dataset) and the evaluation distribution (the distribution that would be induced by the policy π we want to evaluate).

Prior work. There is a substantial body of prior theoretical work that suggests that off-policy evaluation can be difficult and this difficulty scales with some measure of distribution shift. [Wang et al. 2020b; Amortila et al. 2020; Zanette 2021] give exponential (in horizon) lower bounds on sample complexity in the linear setting even with good feature representations that can represent the desired Q function and assuming good data coverage. Upper bounds generally require very strong assumptions on both the representation and limits on the distribution shift [Wang et al. 2021; Duan et al. 2020; Chen and Jiang 2019]. Moreover, the assumed bounds on distribution shift can be exponential in horizon in the worst case. On the empirical side, [Wang et al. 2021] demonstrates issues with distribution shift when learning from pre-trained features and provides a nice discussion of why distribution shift causes error amplification. [Fujimoto et al. 2018a] raises a similar issue under the name "extrapolation error". Regularization and constraints are meant to reduce issues stemming from distribution shift, but also reduce the potential for improvement over the behavior.

Empirical evidence. Both the multi-step and iterative algorithms in our experiments rely on off-policy evaluation as a key subroutine. We examine how easy it is to evaluate the policies encountered along the learning trajectory. To control for issues of iterative error exploitation (discussed in the next subsection), we train Q estimators from scratch on a heldout evaluation dataset sampled from the behavior policy. We then evaluate these trained Q function on rollouts from 1000 datapoints sampled from the replay buffer. Results are shown in Figure 2.3.

The results show a correlation betweed KL and MSE. Moreover, we see that the MSE generally increases over training. One way to mitigate this, as seen in the figure, is to use a large value of α. We just cannot take a very large step before running into problems with distribution shift. But, when we take such a small step, the information from the on-policy $\widehat{Q}^{\beta}$ is about as useful as the newly estimated $\widehat{Q}^{\pi}$. This is seen, for example, in Figure 2.2 where we get very similar performance across algorithms at high levels of regularization.

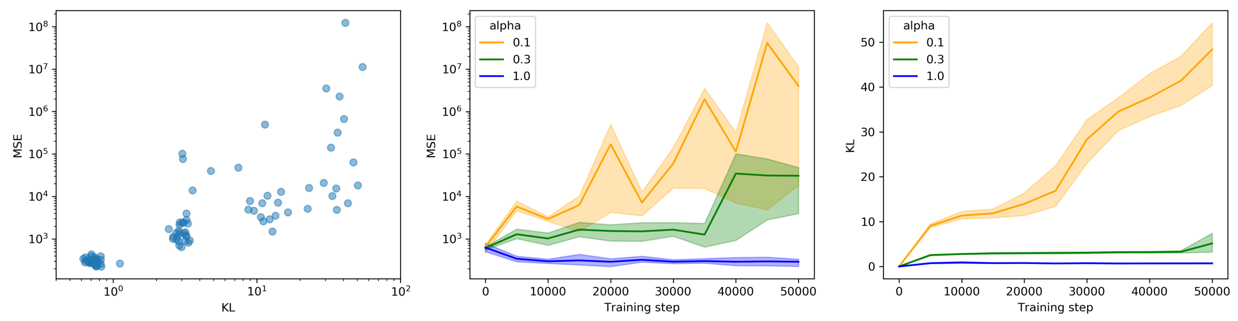


Figure 2.3: Results of running the iterative algorithm on halfcheetah-medium. Each checkpointed policy is evaluated by a Q function trained from scratch on heldout data. MSE refers to $\mathbb{E}_{s,a\sim\beta}[(\hat{Q}^{\pi_i}(s,a)-Q^{\pi_i}(s,a))^2]$ and KL refers to $\mathbb{E}_{s\sim\beta}[KL(\pi(\cdot|s)\|\beta(\cdot|s)]$. Left: 90 policies taken from various points in training with various hyperaparmeters and random seeds. Center: MSE learning curves. Right: KL learning curves. Error bars show min and max over 3 random seeds.

2.6.3 Iterative error exploitation

The previous subsection identifies how any algorithm that uses off-policy evaluation is fundamentally limited by distribution shift, even if we were given fresh data and trained Q functions from scratch at every iteration. But, in practice, iterative algorithms repeatedly iterate between optimizing policies against estimated Q functions and re-estimating the Q functions using the *same data* and using the Q function from the previous step to warm-start the re-estimation. This induces dependence between steps that causes a problem that we call iterative error exploitation.

Intuition about the problem. In short, iterative error exploitation happens because π_i tends to choose overestimated actions in the policy improvement step, and then this overestimation propagates via dynamic programming in the policy evaluation step. To illustrate this issue more formally, consider the following: at each s, a we suffer some Bellman error $\varepsilon^{\pi}_{\beta}(s,a)$ based on our fixed dataset collected by β. Formally,

$$\widehat{Q}^{\pi}(s,a) = r(s,a) + \gamma \mathop{\mathbb{E}}_{\substack{s'|s,a\\ a'\sim\pi|s'}}[\widehat{Q}^{\pi}(s',a')] + \varepsilon^{\pi}_{\beta}(s,a). \tag{2.5}$$

Intuitively, ε_β^π will be larger at state-actions with less coverage in the dataset collected by β. Note that ε_β^π can absorb all error whether it is caused by the finite sample size or function approximation error.

All that is needed to cause iterative error exploitation is that the ε_β^π are highly correlated across different π, but for simplicity, we will assume that ε_β^π is *the same* for all policies π estimated from our fixed offline dataset and instead write ε_β. Now that the errors do not depend on the policy we can treat the errors as auxiliary rewards that obscure the true rewards and see that

$$\widehat{Q}^\pi(s,a) = Q^\pi(s,a) + \widetilde{Q}_\beta^\pi(s,a), \qquad \widetilde{Q}_\beta^\pi(s,a) := \mathop{\mathbb{E}}_{\pi|s_0,a_0=s,a}\left[\sum_{t=0}^{\infty}\gamma^t \varepsilon_\beta(s_t,a_t)\right]. \tag{2.6}$$

This assumption is somewhat reasonable since we expect the error to primarily depend on the data. And, when the prior Q function is used to warm-start the current one (as is generally the case in practice), the approximation errors are automatically passed between steps.

Now we can explain the problem. Recall that under our assumption the ε_β are fixed once we have a dataset and likely to have larger magnitude the further we go from the support of the dataset. So, with each step π_i is able to better maximize ε_β, thus moving further from β and increasing the magnitude of $\widetilde{Q}_\beta^{\pi_i}$ relative to Q^{π_i}. Even though Q^{π_i} may provide better signal than Q^β, it can easily be drowned out by $\widetilde{Q}_\beta^{\pi_i}$. In contrast, $\widetilde{Q}_\beta^\beta$ has small magnitude, so the one-step algorithm is robust to errors[1].

An example. Now we consider a simple gridworld example to illustrate iterative error exploitation. This example fits exactly into the setup outlined above since all errors are due to reward estimation so the ε_β is indeed constant over all π. The gridworld we consider has one determin-

[1]We should note that iterative error exploitation is similar to the overestimation addressed by double Q learning [Van Hasselt et al. 2016; Fujimoto et al. 2018b], but distinct. Since we are in the offline setting, the errors due to our finite dataset can be iteratively exploited more and more, while in the online setting considered by double Q learning, fresh data prevents this issue. We are also considering an algorithm based on policy iteration rather than value iteration.

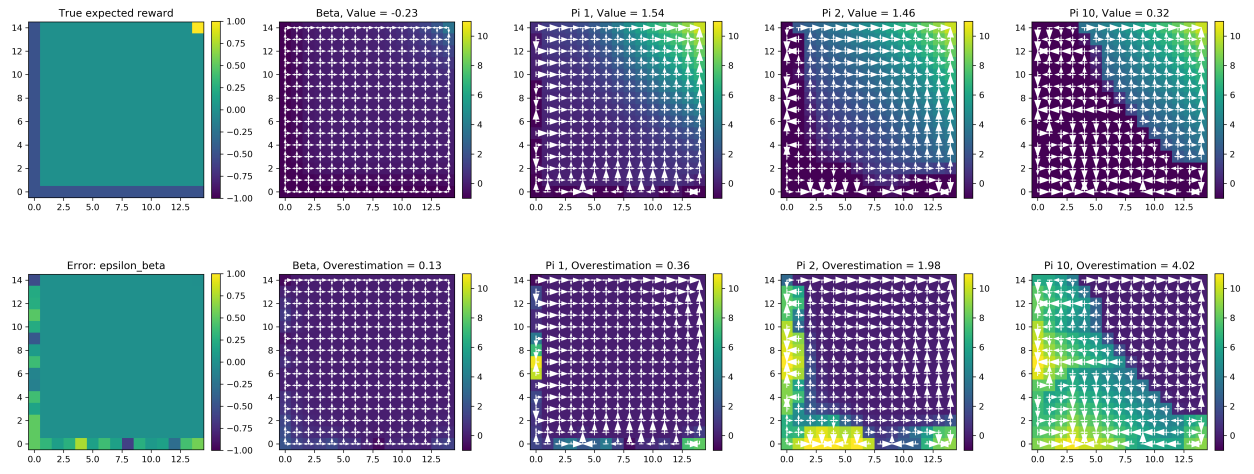


Figure 2.4: An illustration of multi-step offline regularized policy iteration. The leftmost panel in each row shows the true reward (top) or error ε_β (bottom). Then each subsequent panel plots π_i (with arrow size proportional to $\pi_i(a|s)$) over either Q^{π_i} (top) or $\widetilde{Q}^{\pi}_{\beta}$ (bottom), averaged over actions at each state. The one-step policy (π_1) has the highest value. The behavior policy here is a mixture of optimal π^* and uniform u with coefficient 0.2 so that $\beta = 0.2 \cdot \pi^* + 0.8 \cdot u$. We set $\alpha = 0.1$ as the regularization parameter for reverse KL regularization.

istic good state with reward 1 and many stochastic bad states that have rewards distributed as $\mathcal{N}(-0.5, 1)$. We collect a dataset of 100 trajectories, each of length 100. One run of the multi-step offline regularized policy iteration algorithm is illustrated in Figure 2.4.

In the example we see that one step often outperforms multiple steps of improvement. Intuitively, when there are so many noisy states, it is likely that a few of them will be overestimated. Since the data is re-used for each step, these overestimations persist and propagate across the state space due to iterative error exploitation. This property of having many bad, but poorly estimated states likely also exists in the high-dimensional control problems encountered in the benchmark where there are many ways for the robots to fall down that are not observed in the data for non-random behavior. Moreover, both settings have larger errors in areas where we have less data. So even though the errors in the gridworld are caused by noise in the rewards, while errors in D4RL are caused by function approximation, we think this is a useful mental model of the problem.

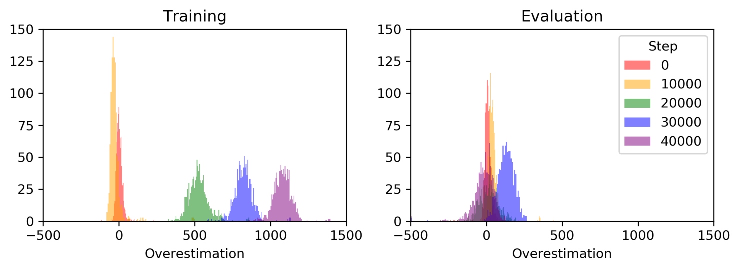


Figure 2.5: Histograms of overestimation error $(\widehat{Q}^{\pi_i}(s,a) - Q^{\pi_i}(s,a))$ on halfcheetah-medium with the iterative algorithm. Left: errors from the training Q function. Right: errors from an independently trained Q function.

Empirical evidence. In practice we cannot easily visualize the progression of errors. However, the dependence between steps still arises as overestimation of the Q values. We can track the overestimation of the Q values over training as a way to measure how much bias is being induced by optimizing against our dependent Q estimators. As a control we can also train Q estimators from scratch on independently sampled evaluation data. These independently trained Q functions do not have the same overestimation bias even though the squared error does tend to increase as the policy moves further from the behavior (as seen in Figure 2.3). Explicitly, we track 1000 state, action pairs from the replay buffer over training. For each checkpointed policy we perform 3 rollouts at each state to get an estimate of the true Q value and compare this to the estimated Q value. Results are shown in Figure 2.5.

2.7 When are multiple steps useful?

So far we have focused on why the one-step algorithm often works better than the multi-step and iterative algorithms. However, we do not want to give the impression that one-step is always better. Indeed, our own experiments in Section 2.5 show a clear advantage for the multi-step and iterative approaches when we have randomly collected data. While we cannot offer a precise delineation of when one-step will outperform multi-step, in this section we offer some intuition

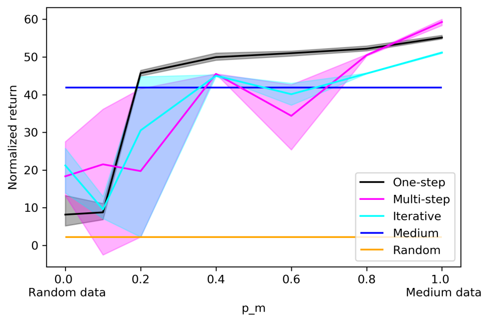


Figure 2.6: Performance of all three algorithms with reverse KL regularization across mixtures between halfcheetah-random and halfcheetah-medium. Error bars indicate min and max over 3 seeds.

as to when we can expect to see benefits from multiple steps of policy improvement.

As seen in Section 2.6, multi-step and iterative algorithms have problems when they propagate estimation errors. This is especially problematic in noisy and/or high dimensional environments. While the multi-step algorithms propagate this noise more widely than the one-step algorithm, they also propagate the signal. So, when we have sufficient coverage to reduce the magnitude of the noise, this increased propagation of signal can be beneficial. The D4RL experiments suggest that we are usually on the side of the tradeoff where the errors are large enough to make one-step preferable.

In Appendix A.1 we illustrate a simple gridworld example where a slight modification of the behavior policy from Figure 2.4 makes multi-step dramatically outperform one-step. This modified behavior policy (1) has better coverage of the noisy states (which reduces error, helping multi-step), and (2) does a worse job propagating the reward from the good state (hurting one-step).

We can also test empirically how the behavior policy effects the tradeoff between error and signal propagation. To do this we construct a simple experiment where we mix data from the random behavior policy with data from the medium behavior policy. Explicitly we construct a dataset D out of the datasets D_r for random and D_m for medium such that each trajectory in D comes from the medium dataset with probability p_m. So for $p_m = 0$ we have the random dataset and $p_m = 1$ we have the medium dataset, and in between we have various mixtures. Results are shown in

Figure 2.6. It takes surprisingly little data from the medium policy for one-step to outperform the iterative algorithm.

2.8 Discussion

This paper presents the surprising effectiveness of a simple one-step baseline for offline RL. We examine the failure modes of iterative algorithms and the conditions where we might expect them to outperform the simple one-step baseline. This provides guidance to a practitioner that the simple one-step baseline is a good place to start when approaching an offline RL problem.

But, we leave many questions unanswered. One main limitation is that we lack a clear theoretical characterization of which environments and behaviors can guarantee that one-step outperforms multi-step or visa versa. Such results will likely require strong assumptions, but could provide useful insight. We don't expect this to be easy as it requires understanding policy iteration which has been notoriously difficult to analyze, often converging much faster than the theory would suggest [Sutton and Barto 2018; Agarwal et al. 2019]. Another limitation is that while only using one step is perhaps the simplest way to avoid the problems of off-policy evaluation, there are possibly other more elaborate algorithmic solutions that we did not consider here. However, our strong empirical results suggest that the one-step algorithm is at least a strong baseline.

3 | Return conditioned supervised learning for offline reinforcement learning

3.1 Introduction

In recent years, deep learning has proven to be an exceptionally powerful generic algorithm for solving supervised learning (SL) tasks. These approaches tend to be stable, and scale well with compute and data [Kaplan et al. 2020]. In contrast, deep reinforcement learning algorithms seem to lack these nice properties; results are well known to be sensitive to hyperparameters and difficult to replicate. In spite of this, deep reinforcement learning (RL) has achieved impressive feats, such as defeating human champions at Go [Silver et al. 2016]. This juxtaposition of success and instability has inspired researchers to explore alternative approaches to reinforcement learning that more closely resemble supervised learning in hopes of making deep RL as well-behaved as deep SL.

One family of algorithms that has garnered great interest recently is return-conditioned supervised learning (RCSL). The core idea of RCSL is to learn the return-conditional distribution of actions in each state, and then define a policy by sampling from the distribution of actions that receive high return. This was first proposed for the online RL setting by work on Upside Down RL [Schmidhuber 2019; Srivastava et al. 2019] and Reward Conditioned Policies [Kumar et al. 2019b]. The idea was extended to the offline RL setting using transformers that condition on the entire history of states rather than just the current Markovian state in the Decision Transformer (DT) work [Chen et al. 2021b; Furuta et al. 2021]. Recent work on RL via Supervised Learning (RvS) [Emmons et al. 2021] unifies and simplifies ideas from these prior works with ideas about goal-conditioned policies.

Importantly, none of this prior work provides theoretical guarantees or analysis of the failure modes of the return-conditioning approach. In contrast, the more established dynamic programming (DP) algorithms for RL are better understood theoretically. This paper attempts to address this gap in understanding, in order to assess when RCSL is a reliable approach for offline RL. Specifically, we answer the following questions:

- What optimality guarantees can we make for RCSL? Under what conditions are they necessary and sufficient?
- In what situations does RCSL fail in theory and in practice?
- How does RCSL relate to other approaches, such as DP and behavior cloning (BC)?

We find that although RCSL does select a near-optimal policy under certain conditions, the necessary assumptions are more strict than those for DP. In particular, RCSL (but not DP) requires nearly deterministic dynamics in the MDP, knowledge of the proper value to condition on, and for the conditioning value to be supported by the distribution of returns in the dataset. We provide simple tabular examples to demonstrate the necessity of these assumptions. The shortcomings of RCSL that we identify in theory are verified empirically with some simple experiments using neural models on ad-hoc example problems as well as benchmark datasets. We conclude that RCSL alone is unlikely to be a general solution for offline RL problems, but does show promise in some specific situations such as deterministic MDPs with high-quality behavior data.

3.2 Preliminaries

3.2.1 Setup

We will consider an offline RL setup where we are given a dataset $\mathcal{D}$ of trajectories τ where each $\tau = (o_1, a_1, r_1, \cdots, o_H, a_H, r_H)$ contains observations $o_t \in \mathcal{O}$, actions $a_t \in \mathcal{A}$, and rewards $r_t \in [0,1]$ generated by some behavior policy β interacting with a finite horizon MDP with horizon H. Let $g(\tau) = \sum_{t=1}^{H} r_t$ denote the cumulative return of the trajectory (we will just use g when the trajectory is clear from context). And let $J(\pi) = \mathbb{E}_{\tau\sim\pi}[g(\tau)]$ be the expected return of a policy π. We then let the state representation $s_t \in \mathcal{S}$ be any function of the history of observations, actions, and rewards up to step t along with o_t. To simplify notation in the finite horizon setting, we will sometimes drop the timestep from s to refer to generic states and assume that we can access the timestep from the state representation as $t(s)$. Let P_π denote the joint distribution over states, actions, rewards, and returns induced by any policy π.

In this paper, we focus on the RCSL approach that learns by return-conditioned supervised learning. Explicitly, at training time this method minimizes the empirical negative log likelihood loss:

$$\hat{L}(\pi) = -\sum_{\tau\in\mathcal{D}} \sum_{1\le t\le H} \log \pi(a_t|s_t, g(\tau)). \tag{3.1}$$

Then at test time, an algorithm takes the learned policy π along with a conditioning function $f(s)$ to define the test-time policy π_f as:

$$\pi_f(a|s) := \pi(a|s, f(s)). \tag{3.2}$$

Nota bene: the Decision Transformer [Chen et al. 2021b] is captured in this framework by defining the state space so that the state s_t at time t also contains all past $o_{t'}$, $a_{t'}$, and $r_{t'}$ for $t' < t$. In prior work, f is usually chosen to be a constant at the initial state and to decrease with observed reward along a trajectory, which is captured by a state representation that includes the history of rewards.

3.2.2 The RCSL policy

To better understand the objective, it is useful to first consider its optimum in the case of infinite data. It is clear that our loss function attempts to learn $P_\beta(a|s,g)$ where β is the behavior policy that generated the data (and recall that P_β refers to the distribution over states, actions, and returns induced by β). Factoring this distribution, we quickly see that the optimal policy π_f^{RCSL} for a specific conditioning function f can be written as:

$$\pi_f^{\text{RCSL}}(a|s) = P_\beta(a|s, f(s)) = \frac{P_\beta(a|s)P_\beta(f(s)|s,a)}{P_\beta(f(s)|s)} = \beta(a|s)\frac{P_\beta(f(s)|s,a)}{P_\beta(f(s)|s)}. \tag{3.3}$$

Essentially, the RCSL policy re-weights the behavior based on the distribution of future returns.

Connection to distributional RL. In distributional RL [Bellemare et al. 2022], the distribution of future returns under a policy π from state s and action a is defined as: $G^\pi(s,a) \sim g = \sum_{t=t(s)}^{H} r_t \mid \tau \sim \pi, s_{t(s)} = s, a_{t(s)} = a$. The RCSL policy is precisely proportional to the product of the behavior policy and the density of the distributional Q function of the behavior policy (i.e. $P_\beta(g|s,a)$).

3.2.3 Related work

As noted in the introduction, our work is in direct response to the recent line of literature on RCSL [Schmidhuber 2019; Srivastava et al. 2019; Kumar et al. 2019b; Chen et al. 2021b; Furuta et al. 2021; Emmons et al. 2021]. Specifically, we will focus on the DT [Chen et al. 2021b] and RvS [Emmons et al. 2021] formulations in our experiments since they also focus on the offline RL setting. Note that another recent work introduced the Trajectory Transformer [Janner et al. 2021] which does not fall under the RCSL umbrella since it performs planning in the learned model to define a policy.

Another relevant predecessor of RCSL comes from work on goal-based RL [Kaelbling 1993]. Compared to RCSL, this line of work replaces the target return g in the empirical loss function by a goal state. One instantiation is hindsight experience replay (HER) where each trajectory in the replay buffer is relabeled as if the observed final state was in fact the goal state [Andrychowicz et al. 2017]. Another instance is goal-conditioned supervised learning [GCSL, Ghosh et al. 2019], which provides more careful analysis and guarantees, but the guarantees (1) are not transferable to the return-conditioned setting, (2) assume bounds on L_∞ errors in TV distance instead of dealing with expected loss functions that can be estimated from data, and (3) do not provide analysis of the tightness of the bounds.

Concurrent work [Štrupl et al. 2022; Paster et al. 2022; Yang et al. 2022] also all raise the issue of RCSL in stochastic environments with infinite data, and present some algorithmic solutions. However, none of this work addresses the potentially more fundamental issue of sample complexity that arises from the requirement of return coverage that we discuss in Section 4.

3.3 When does RCSL find the optimal policy?

We begin by exploring how RCSL behaves with infinite data and a fully expressive policy class. In this setting, classic DP algorithms (e.g. Q-learning) are guaranteed to converge to the optimal policy under coverage assumptions [Sutton and Barto 2018]. But we now show that this is not the case for RCSL, which requires additional assumptions for a similar guarantee. Our approach is to first derive a positive result: under certain assumptions, the policy which optimizes the RCSL objective (Section 3.2.2) is guaranteed to be near-optimal. We then illustrate the limitations of

RCSL by providing simple examples that are nonetheless challenging for these methods in order to demonstrate why our assumptions are necessary and that our bound is tight.

Theorem 3.1 (Alignment with respect to the conditioning function)**.** *Consider an MDP, behavior β and conditioning function f. Assume the following:*

1. *Return coverage: $P_\beta(g = f(s_1)|s_1) \geq \alpha_f$ for all initial states s_1.*
2. *Near determinism: $P(r \neq r(s,a)$ or $s' \neq T(s,a)|s,a) \leq \epsilon$ at all s, a for some functions T and r. Note that this does not constrain the stochasticity of the initial state.*
3. *Consistency of f: $f(s) = f(s') + r$ for all s.*[1]

Then

$$\mathop{\mathbb{E}}_{s_1}[f(s_1)] - J(\pi_f^{RCSL}) \leq \epsilon \left(\frac{1}{\alpha_f} + 2 \right) H^2. \tag{3.4}$$

Moreover, there exist problems where the bound is tight up to constant factors.

The proof is in Appendix B.3.1. Note that the quantity $\mathbb{E}_{s_1}[f(s_1)]$ is specific to the structure of RCSL algorithms and captures the notion that the ideal RCSL policy will be able to reproduce policies of any value when given different conditioning functions (with appropriate data). The theorem immediately yields the following corollaries (with proof in Appendix B.3.1).

Corollary 3.2. *Under the assumptions of Theorem 3.1, there exists a conditioning function f such that*

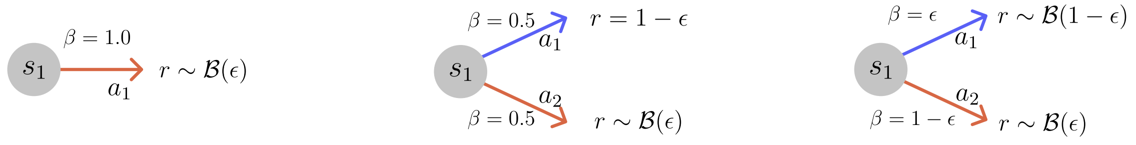
$$J(\pi^*) - J(\pi_f^{RCSL}) \leq \epsilon \left(\frac{1}{\alpha_f} + 3 \right) H^2. \tag{3.5}$$

[1]Note this can be exactly enforced (as in prior work) by augmenting the state space to include the cumulative reward observed so far.

Corollary 3.3. *If $\alpha_f > 0$, $\epsilon = 0$, and $f(s_1) = V^*(s_1)$ for all initial states s_1, then $J(\pi_f^{RCSL}) = J(\pi^*)$.*

The corollaries tell us that in near determinisitc environments with the proper conditioning functions and data coverage, it is possible for RCSL to recover near optimal policies. These assumptions are somewhat strong compared to those needed for DP-based approaches, so we will now explain why they are necessary for our analysis.

Tightness. To demonstrate tightness we will consider the simple examples in Figure 3.1. These MDPs and behavior policies demonstrate tightness in ϵ and α_f up to constant factors, and provide insight into how stochastic dynamics lead to suboptimal behavior from RCSL algorithms.



(a) An example where the bound is tight. $\mathcal{B}$ denotes the Bernoulli distribution.

(b) An example where RCSL also has large regret.

(c) An example where RCSL also has large regret for any conditioning function.

Figure 3.1: Failure modes of RCSL in stochastic environments with infinite data.

First, consider the example in Figure 3.1(a) with conditioning $f(s_1) = 1$. There is only one possible policy in this case, and it has $J(\pi) = \epsilon$ so that $\mathbb{E}[f(s_1)] - J(\pi) = 1 - \epsilon$. Note that $\alpha_f = \epsilon$, so we have that $\epsilon/\alpha_f = 1$. Thus, the bound is tight in ϵ/α_f. This example shows that the goal of achieving a specific desired return is incompatible with stochastic environments.

This first example is somewhat silly since there is only one action, so the learned policy does not actually suffer any regret. To show that this issue can in fact lead to regret, consider the example in Figure 3.1(b), again with conditioning $f(s_1) = 1$. Then applying the reasoning from Section 3.2.2,

$$\pi_f^{\text{RCSL}}(a_1|s_1) = \beta(a_1|s_1)\frac{P_\beta(g = 1|s_1, a_1)}{P_\beta(g = 1|s_1)} = 0.5 \cdot \frac{0}{0.5 \cdot \epsilon} = 0. \tag{3.6}$$

So we get that $\mathbb{E}[f(s_1)] - J(\pi_f^{\text{RCSL}}) = 1 - \epsilon$, while $\epsilon/\alpha_f = \epsilon/(\epsilon/2) = 2$ (which is on the same order, up to a constant factor). However, in this case the learned policy π_f^{RCSL} suffers substantial regret since the chosen action a_2 has substantially lower expected value than a_1 by $1 - 2\epsilon$.

The issue in the second example could be resolved by changing the conditioning function so that $f(s_1) = 1 - \epsilon$. Now we will consider the example in Figure 3.1(c) where we will see that there exist cases where the bias of RCSL in stochastic environments can remain regardless of the conditioning function. In this MDP, the only returns that are supported are $g = 0$ or $g = 1$. For $f(s_1) = 1$, plugging in to the formula for π_f yields

$$\pi_f^{\text{RCSL}}(a_1|s_1) = \beta(a_1|s_1)\frac{P_\beta(g=1|s_1,a_1)}{P_\beta(g=1|s_1)} = \epsilon\frac{1-\epsilon}{\epsilon(1-\epsilon)+(1-\epsilon)\epsilon} = \frac{1}{2}. \tag{3.7}$$

Thus, $\mathbb{E}[f(s_1)] - J(\pi_f^{\text{RCSL}}) = 1/2$ and $J(\pi^*) - J(\pi_f^{\text{RCSL}}) = 1/2 - \epsilon$. This shows that merely changing the conditioning function is not enough to overcome the bias of the RCSL method in stochastic environments.

These examples show that even for MDPs that are ϵ-close to being deterministic, the regret of RCSL can be large. But, in the special case of deterministic MDPs we find that RCSL can indeed recover the optimal policy. And note that we still allow for stochasticity in the initial states in these deterministic MDPs, which provides a rich setting for problems like robotics that requires generalization over the state space from finite data. In the next section, we will consider more precisely what happens to RCSL algorithms with finite data and limited model classes.

Trajectory stitching. Another issue often discussed in the offline RL literature is the idea of trajectory stitching [Wang et al. 2020c; Chen et al. 2021b]. Ideally, an offline RL agent can take suboptimal trajectories that overlap and stitch them into a better policy. Clearly, DP-based algorithms can do this, but it is not so clear that RCSL algorithms can. In Appendix B.2 we provide theoretical and empirical evidence that in fact they cannot perform stitching in general, even with infinite data. While this does not directly affect our bounds, the failure to perform stitching is an issue of practical importance for RCSL methods.

3.4 Sample complexity of RCSL

Now that we have a better sense of what policy RCSL will converge to with infinite data, we can consider how quickly (and under what conditions) it will converge to the policy π_f when given finite data and a limited policy class, as will occur in practice. We will do this via a reduction from the regret relative to the infinite data solution π_f to the expected loss function L minimized at training time by RCSL, which is encoded in the following theorem.

Theorem 3.4 (Reduction of RCSL to SL)**.** *Consider any function $f : \mathcal{S} \to \mathbb{R}$ such that the following two assumptions hold:*

1. *Bounded occupancy mismatch: $\frac{P_{\pi_f^{RCSL}}(s)}{P_\beta(s)} \leq C_f$ for all s.*
2. *Return coverage: $P_\beta(g = f(s)|s) \geq \alpha_f$ for all s.*

Define the expected loss as $L(\hat{\pi}) = \mathbb{E}_{s\sim P_\beta} \mathbb{E}_{g\sim P_\beta(\cdot|s)} [D_{\mathrm{KL}}(P_\beta(\cdot|s, g) \| \hat{\pi}(\cdot|s, g))]$. Then for any estimated RCSL policy $\hat{\pi}$ that conditions on f at test time (denoted by $\hat{\pi}_f$), we have that

$$J(\pi_f^{RCSL}) - J(\hat{\pi}_f) \leq \frac{C_f}{\alpha_f} H^2 \sqrt{2L(\hat{\pi})}. \tag{3.8}$$

The proof can be found in Appendix B.3.3. Note that we require a similar assumption of return coverage as before to ensure we have sufficient data to define π_f. We also require an assumption on the state occupancy of the idealized policy π_f relative to β. This assumption is needed since the loss $L(\hat{\pi})$ is optimized on states sampled from P_β, but we care about the expected return of the learned policy relative to that of π_f, which can be written as an expectation over states sampled from P_{π_f}.

This gives us a reduction to supervised learning, but to take this all the way to a sample complexity bound we need to control the loss $L(\hat{\pi})$ from finite samples. Letting N denote the size of the dataset, the following corollary uses standard uniform convergence results from supervised learning [Shalev-Shwartz and Ben-David 2014] to yield finite sample bounds.

Corollary 3.5 (Sample complexity of RCSL)**.** *To get finite data guarantees, add to the above assumptions the assumptions that (1) the policy class* Π *is finite, (2)* $|\log \pi(a|s,g) - \log \pi(a'|s',g')| \leq c$ *for any* (a,s,g,a',s',g') *and all* $\pi \in \Pi$*, and (3) the approximation error of* Π *is bounded by* ϵ_{approx}*, i.e.* $\min_{\pi\in\Pi} L(\pi) \leq \epsilon_{approx}$*. Then with probability at least* $1-\delta$*,*

$$J(\pi_f^{RCSL}) - J(\hat{\pi}_f) \leq O\left(\frac{C_f}{\alpha_f} H^2 \left(\sqrt{c}\left(\frac{\log|\Pi|/\delta}{N}\right)^{1/4} + \sqrt{\epsilon_{approx}}\right)\right). \tag{3.9}$$

The proof is in Appendix B.3.4. Analyzing the bound, we can see that the dependence on N is in terms of a fourth root rather than the square root, but this comes from the fact that we are optimizing a surrogate loss. Namely the learner optimizes KL divergence, but we ultimately care about regret which we access by using the KL term to bound a TV divergence and thus lose a square root factor. A similar rate appears, for example, when bounding 0-1 classification loss of logistic regression [Bartlett et al. 2006; Boucheron et al. 2005].

This corollary also tells us something about how the learner will learn to generalize across different values of the return. If the policy class is small (for some notion of model complexity) and sufficiently structured, then it can use information from the given data to generalize across values of g, using low-return trajectories to inform the model on high-return trajectories.

Note that a full sample complexity bound that competes with the optimal policy can be derived by combining this result with Corollary 3.2 as follows:

Corollary 3.6 (Sample complexity against the optimal policy)**.** *Under all of the assumptions of*

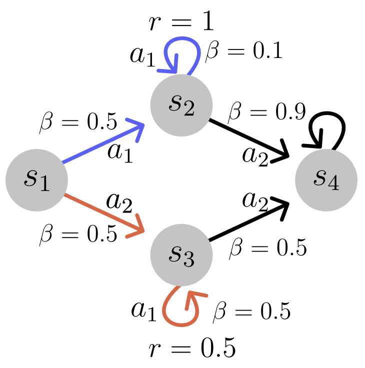


Figure 3.2: An example where RCSL has exponential sample complexity in a deterministic environment.

Corollary 3.2 and Corollary 3.5 we get:

$$J(\pi^*) - J(\hat{\pi}_f) \leq O\left(\frac{C_f}{\alpha_f} H^2 \left(\sqrt{c}\left(\frac{\log|\Pi|/\delta}{N}\right)^{1/4} + \sqrt{\epsilon_{approx}}\right) + \frac{\epsilon}{\alpha_f} H^2\right). \tag{3.10}$$

Tightness. To better understand why the dependence on $1/\alpha_f$ is tight and potentially exponential in the horizon H, even in deterministic environments, we offer the example in Figure 3.2. Specifically, we claim that any value of $f(s_1)$ where the policy π_f^{RCSL} prefers the good action a_1 from s_1 will require on the order of $10^{H/2}$ samples in expectation to recover as $\hat{\pi}_f$[2].

To see why this is the case, we consider the MDP illustrated in Figure 3.2 with horizon $H \gg 4$. The MDP has four states each with two actions. All transitions and rewards are deterministic. The only actions with non-zero reward are $r(s_2, a_1) = 1$ and $r(s_3, a_1) = 0.5$. The interesting decision is at s_1 where a_1 is better than a_2.

Note that for any integer $1 \leq k < H/2$, we have that $P_\beta(g = k|s_1, a_2) = 0.5 \cdot 0.5^{2k} = 0.5 \cdot (0.25)^k$, while $P_\beta(g = k|s_1, a_1) = 0.5 \cdot (0.1)^k$. Conditioning on any such k will make us more likely to choose the bad action a_2 from s_1. The only way to increase the likelihood of the good action a_1 from s_1 and s_2 is to condition on $f(s_1) > H/2$. Unfortunately for RCSL, the probability of observing $g > H/2$ is extremely small, since for any such f we have $P_\beta(g = f(s_1)) \leq 0.5 \cdot (0.1)^{H/2} \leq 10^{-H/2}$.

[2]Except for $f(s_1) = 0$, which will yield a policy substantially worse than the behavior.

Thus, both α_f and the sample complexity of learning for any f that will yield a policy better than the behavior is exponential in the horizon H.

Fundamentally, the problem here is that RCSL uses trajectory-level information instead of performing dynamic programming on individual transitions. But, collecting enough trajectory-level information can take exponentially many samples in the horizon. In contrast, DP merely requires coverage of transitions in the MDP to perform planning and thus avoids this issue of exponential sample complexity. In the next section we will delve deeper into this comparison with DP-based approaches as well as the simple top-% BC baseline.

3.5 Comparing RCSL with bounds for alternative methods

Now that we understand the rate at which we expect RCSL to converge, we briefly present the convergence rates of two baseline methods for comparison. In particular, we will leverage an existing analysis of a DP-based algorithm, and conduct a novel analysis of top-% BC. We find that the sample complexity of RCSL has a similar rate to top-% BC, and is worse than DP due to the potentially exponential dependence on horizon that stems from return coverage.

3.5.1 Comparison to dynamic programming.

We will compare to the state of the art (to our knowledge) bound for a DP-based offline RL algorithm. Namely, we will look at the results of [Xie et al. 2021] for pessimistic soft policy iteration. Similar results exist for slightly different algorithms or assumptions [Chen and Jiang 2019; Wang et al. 2020b], but we choose this one since it is both the tightest and more closely aligns with the practical actor-critic algorithms that we use for our experiments. Their bound makes the following assumptions about the function class F and the dataset (letting $\mathcal{T}^\pi$ represent the Bellman operator for policy π):

1. Realizability: for any policies π, π' there exists $f \in F$ with $\|f - \mathcal{T}^\pi f\|_{2,P_{\pi'}}^2 \leq \epsilon_1$.

2. Bellman completeness: for any π and $f \in F$ there exists $f' \in F$ such that $\|f' - \mathcal{T}^\pi f\|_{2,P_\beta}^2 \leq \epsilon_2$.

3. Coverage: $\frac{P_{\pi^*}(s,a)}{P_\beta(s,a)} \leq C$ for all s, a[3].

With these assumptions in place, the sample complexity bound takes the form[4]:

$$J(\pi^*) - J(\hat{\pi}) \leq O\left(H^2\left(\sqrt{\frac{C \log |F||\Pi|/\delta}{N}}\right) + H^2\sqrt{C(\epsilon_1 + \epsilon_2)}\right) \tag{3.11}$$

Note: this is the result for the "information-theoretic" form of the algorithm that cannot be efficiently implemented. The paper also provides a "practical" version of the algorithm for which the bound is the same except that the the square root in the first term is replaced with a fifth root.

There are several points of comparison with our analysis (specifically, our Corollary 3.6). The first thing to note is that for RCSL to compete with the optimal policy, we require nearly deterministic dynamics and a priori knowledge of the optimal conditioning function. These assumptions are not required for the DP-based algorithm; this is a critical difference, since it is clear that these conditions often do not hold in practice.

Comparing the coverage assumptions, our C_f becomes nearly equivalent to C. The major difference is that our analysis of RCSL also requires dependence on return coverage $1/\alpha_f$. This is problematic since as seen in Section 3.4, this return coverage dependence can be exponential in horizon in cases where the state coverage does not depend on horizon.

Comparing the approximation error assumptions, we see that the realizability and completeness assumptions required for DP are substantially less intuitive than the standard supervised learning approximation error assumption needed for RCSL. These assumptions are not directly comparable, but intuitively the RCSL approximation error assumption is simpler.

[3]The original paper uses a slightly tighter notion of coverage, but this bound will suffice for our comparison.

[4]The original paper considers an infinite horizon discounted setting. For the purposes of comparison, we will just assume that $\frac{1}{1-\gamma}$ can be replaced by H.

Finally, dependence on H is the same for both methods and dependence on N depends on which version of the DP algorithm we compare to. For the information-theoretic algorithm DP has better dependence on N, but for the practical algorithm RCSL has better dependence. It is not clear whether the dependence on N in either the RCSL analysis or in the analysis of the practical algorithm from [Xie et al. 2021] is tight, and it is an interesting direction for future work to resolve this issue.

3.5.2 Comparison to top-% behavior cloning.

The closest algorithm to RCSL is top-% BC, which was introduced as a baseline for Decision Transformers [Chen et al. 2021b]. This algorithm simply sorts all trajectories in the dataset by return and takes the top ρ fraction of trajectories to run behavior cloning (for $\rho \in [0, 1]$). The most obvious difference between this algorithm and RCSL is that RCSL allows us to plug in different conditioning functions at test time to produce different policies, while top-% BC learns only one policy. However, if we want to achieve high returns, the two algorithms are quite similar.

The full statements and proofs of our theoretical results for top-% BC are deferred to Appendix B.3.5. The results are essentially the same as those for RCSL except for two key modifications:

Defining coverage. The first difference in the analysis is the notion of coverage. For RCSL we needed the return distribution to cover the conditioning function f. For top-% BC we instead let g_ρ be the $1 - \rho$ quantile of the return distribution over trajectories sampled by the behavior β and then define coverage as $P_\beta(g \geq g_\rho | s) \geq \alpha_\rho$ for all s. This modification is somewhat minor.

Sample size and generalization. The main difference between RCSL and top-% BC is that the RCSL algorithm attempts to transfer information gained from low-return trajectories while the top-% BC algorithm simply throws those trajectories away. This shows up in the formal bounds since for a dataset of size N the top-% BC algorithm only uses $\rho \cdot N$ samples while RCSL uses all N. Depending on the data distribution, competing with the optimal policy may require setting ρ very close to zero (exponentially small in H) yielding poor sample complexity.

These bounds suggest that RCSL can use generalization across returns to provide improvements in sample complexity over top-% BC by leveraging all of the data. However, the RCSL model is attempting to learn a richer class of functions that conditions on reward, which may require a larger policy class negating some of this benefit. Overall, RCSL should expect to beat top-% BC if the behavior policy is still providing useful information about how to interact with the environment in low-return trajectories that top-% BC would throw away.

3.6 Experiments

We have illustrated through theory and some simple examples when we expect RCSL to work, but the theory does not cover all cases that are relevant for practice. In particular, it is not clear how the neural networks trained in practice can leverage generalization across returns. Moreover, one of the key benefits to RCSL approaches (as compared to DP) is that by avoiding the instabilities of non-linear off-policy DP in favor of supervised learning, one might hope that RCSL is more stable in practice. In this section we attempt to test these capabilities first through targeted experiments in a point-mass environment and then by comparisons on standard benchmark data.

Throughout this section we will consider six algorithms, two from each of three categories:

1. Behavior cloning (BC): standard behavior cloning (BC) and percentage behavior cloning (%BC) that runs BC on the trajectories with the highest returns [Chen et al. 2021b].

2. Dynamic programming (DP): TD3+BC [Fujimoto and Gu 2021] a simple DP-based offline RL approach and IQL [Kostrikov et al. 2021a] a more stable DP-based offline RL approach.

3. Return-conditioned supervised learning (RCSL): RvS [Emmons et al. 2021] an RCSL approach using simple MLP policies, and DT [Chen et al. 2021b] an RCSL approach using transformer policies.

All algorithms are implemented in JAX [Bradbury et al. 2018] using flax [Heek et al. 2020] and the jaxrl framework [Kostrikov 2021], except for DT which is taken from the original paper. Full details can be found in Appendix B.4 and code can be found at `https://github.com/davidbrandfonbrener/rcsl-paper`.

3.6.1 Point-mass datasets

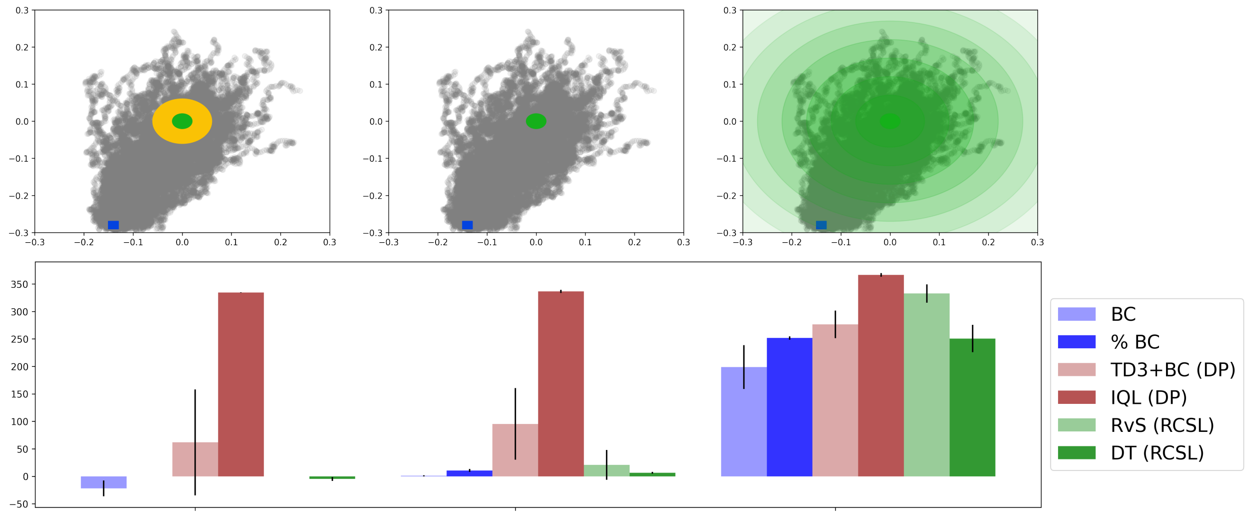
First, we use targeted experiments to demonstrate how the tabular failure modes illustrated above can arise even in simple deterministic MDPs that may be encountered in continuous control. Specifically, we will focus on the issue of exponential sample complexity discussed in Section 3.4. We build our datasets in an environment using the Deepmind control suite [Tassa et al. 2018] and MuJoCo simulator [Todorov et al. 2012a]. The environment consists of a point-mass navigating in a 2-d plane.

To build an example with exponential sample complexity we construct a navigation task with a goal region in the center of the environment. The dataset is constructed by running a behavior policy that is a random walk that is biased towards the top right of the environment. To construct different levels of reward coverage, we consider the environment and dataset under three different reward functions (ordered by probability of seeing a trajectory with high return, from lowest to highest):

(a) The "ring of fire" reward. This reward is 1 within the goal region, -1 in the ring of fire region surrounding the goal, and 0 otherwise

(b) The sparse reward. This reward is 1 within the goal region and 0 otherwise.

(c) The dense reward. This reward function is 1 within the goal region and gradually decays with the Euclidean distance outside of it.



(a) Ring of fire (b) Sparse (c) Dense

Figure 3.3: RCSL fails under reward functions that lead to exponentially small probability of sampling good trajectories, but can generalize when the reward is dense. Error bars show standard deviation across three seeds. BC methods are in blue, DP methods in brown, and RCSL methods in green.

Intuitively, the ring of fire reward will cause serious problems for RCSL approaches when combined with the random walk behavior policy. The issue is that any random walk which reached the goal region is highly likely to spend more time in the region of negative rewards than in the actual goal states, since the ring of fire has larger area than the goal. As a result, while they are technically supported by the distribution, it is unlikely to find many trajectories (if any at all) with positive returns in the dataset, let alone near-optimal returns. As a result, the RCSL-based approaches are not even able to learn to achieve positive returns, as seen in Figure 3.3.

The sparse reward is also difficult for the RCSL-based algorithms, for similar reasons; however the problem is less extreme since any trajectory that gets positive reward must go to the goal, so there is signal in the returns indicating where the goal is. In contrast, the dense reward provides enough signal in the returns that RCSL approaches are able to perform well, although still not as well as IQL. It is also worth noting that because the datset still does not have full coverage of the

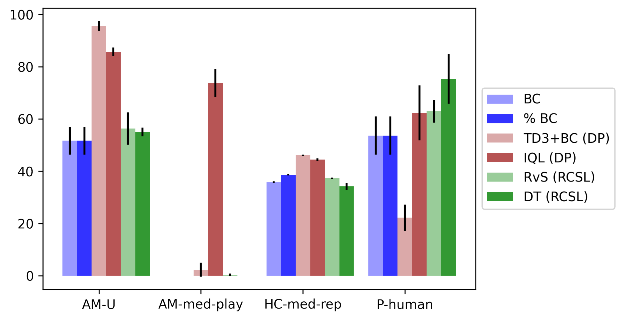


Figure 3.4: Data from ANTMAZE-UMAZE, ANTMAZE-MEDIUM-PLAY, HALFCHEETAH-MEDIUM-REPLAY, and PEN-HUMAN. Error bars show standard deviation across three seeds. Each algorithm is tuned over 4 values and best performance is reported.

state-space, simple DP-based algorithms like TD3+BC can struggle with training instability.

3.6.2 Benchmark data

In addition to our targeted experiments we also ran our candidate algorithms on some datasets from the D4RL benchmark [Fu et al. 2020]. These are meant to provide more realistic and larger-scale data scenarios. While this also makes these experiments less targeted, we can still see that the insights that we gained in simpler problems can be useful in these larger settings. We attempt to choose a subset of the datasets with very different properties from eachother. For example, the play data on the ant-maze environment is very diverse and plentiful while the human demonstration data on the pen environment has poor coverage but high values. Results are shown in Figure 3.4. And additional results leading to similar conclusions can be found in Appendix B.1.

We find that for most of the datasets DP-based algorithms TD3+BC and IQL outperform both the BC-based algorithms and RCSL-based algorithms. This is especially stark on the ANTMAZE datasets where the behavior policy is highly stochastic, requiring the learner to stitch together trajectories to achieve good performance. While none of these tasks has stochastic dynamics, the issues of return coverage and trajectory stitching persist.

In contrast, RCSL performs well when the behavior policy is already high quality, but not optimal (as in the PEN-HUMAN task). Since the data is suboptimal and reward is dense, there is opportunity for RCSL to outperform the BC-based methods. Moreover, since the data has poor coverage, standard DP approaches like TD3+BC are highly unstable. IQL is more stable and performs similarly to the RCSL-based algorithms, but is outperformed by DT (perhaps due to the use of history-dependent policies).

3.7 Discussion

Looking back at our results, we can better place RCSL in relation to the more classical BC and DP algorithms. Like BC, RCSL relies on supervised learning and thus inherits its simplicity, elegance, and ease of implementation and debugging. However, it also inherits BC's dependence on the quality of the behavior policy. This dependence can be somewhat reduced in (nearly) deterministic environments, where conditioning on high returns can break the bias towards the behavior policy. But, the reliance on trajectory-level information still means that RCSL is fundamentally limited by the quality of the best trajectories in the dataset, which can require a sample complexity exponential in horizon in order to compete with the optimal policy, even in deterministic environments.

In contrast, DP methods are capable of learning good policies even when the dataset does not contain any high-return trajectories and the environment is stochastic. This represents a fundamental gap between the two approaches that cannot be bridged within the RCSL paradigm. However, empirically, current deep DP algorithms are not well-behaved. These algorithms are often unstable and difficult to debug, although recent work has started to alleviate these issues somewhat [Kostrikov et al. 2021a].

In sum, for tasks where the requirements for RCSL to perform well are met, it is an excellent practical choice, with great advantages in simplicity over DP. Since many real-world tasks of relevance have these attributes, RCSL techniques could have substantial impact. But as a general learning paradigm, RCSL is fundamentally limited in ways that DP is not.

Part II

Data collection

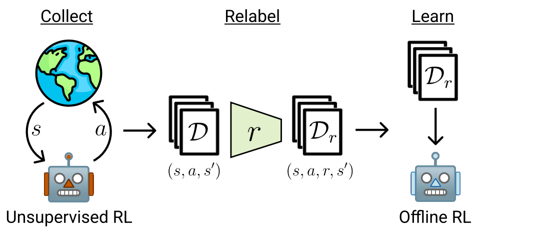
4 Exploratory data for offline reinforcement learning

4.1 Introduction

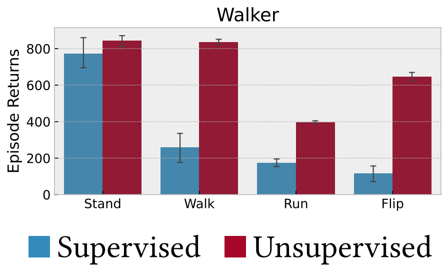
Large and diverse datasets have been the cornerstones of many impressive successes at the frontier of machine learning, such as image recognition [Deng et al. 2009; Krizhevsky et al. 2012; Radford et al. 2021b], natural language processing [Radford et al. 2019; Raffel et al. 2019; Wang et al. 2019; Brown et al. 2020], and protein folding [Consortium 2020; Jumper et al. 2021]. Training on diverse datasets often dramatically improves performance and enables impressive generalization capabilities.

In reinforcement learning (RL), however, the potential of large and diverse datasets has not yet been fully realized. The historically dominant approach in RL has been to train online from scratch by interacting with a task-specific environment [Sutton and Barto 2018; Mnih et al. 2013]. More recently there has been increased interest in learning offline from fixed datasets [Levine et al. 2020; Fujimoto et al. 2019b]. This line of offline RL research has been accelerated by benchmark datasets such as D4RL [Fu et al. 2020] and RL Unplugged [Gulcehre et al. 2020].

Currently, offline RL datasets are collected by training task-specific RL algorithms and saving the data visited during the training process. Offline RL algorithms are then evaluated on the same



(a) The three phases of ExORL: data collection by unsupervised RL, data relabeling by the downstream reward, and learning on the labeled data by offline RL.



(b) Offline RL evaluation using vanilla TD3 of supervised data (the replay buffer of a TD3 agent trained on Walker Stand) versus unsupervised exploratory data (collected reward-free on Walker with Proto).

Figure 4.1: Summary of the ExORL framework.

task that the dataset was created with. As a result, very strong performance on these benchmarks can often be achieved with simple methods like behavior cloning [Pomerleau 1988; Chen et al. 2021c] or one step of policy iteration [Brandfonbrener et al. 2021; Gulcehre et al. 2021] without requiring substantial amounts of planning. This raises a fundamental question for offline RL – What happens when your data does not come from the same task?

In this work, we propose a new framework for offline RL that can test the ability of offline RL algorithms to generalize to tasks that are not observed during the creation of the dataset. This framework, which we call Exploratory data for Offline RL (ExORL), has three steps: collect, relabel, and learn. To instantiate ExORL, we *collect* datasets using nine different unsupervised exploration strategies [Laskin et al. 2021] on several environments from the DeepMind control suite [Tassa et al. 2018]. Then, we *relabel* the data with different reward functions. Finally, we *learn* a policy by running several different offline RL algorithms for each relabeled dataset. A schematic of the framework can be found in Figure 4.1(a) and an instantiation in a simple pointmass maze environment in Figure 4.2.

We provide a thorough analysis of current offline RL methods by running large-scale experiments with ExORL. The main findings of these experiments are:

1. With sufficiently diverse exploratory data, vanilla off-policy RL agents can effectively learn

offline, and even outperform carefully designed offline RL algorithms. This suggests that advances in data collection are as important as algorithmic advances for offline RL.

2. Compared to data collected in a supervised manner (i.e. for a task-specific reward function), the unsupervised exploratory data allows offline RL algorithms to learn multiple behaviors from a single ExORL dataset. For example, unsupervised exploration yields a 106% performance increase over supervised data on multi-task learning in the Walker environment (Figure 4.1(b)).

4.2 Related Work

4.2.1 Datasets for offline RL

The most closely related line of work is that of data-collection and benchmarking in offline RL. Both the D4RL [Fu et al. 2020] and RL Unplugged [Gulcehre et al. 2020; Mathieu et al. 2021] benchmark suites consist of datasets collected by policies attempting to optimize task-specific rewards. These datasets are either replay buffers of the training run or trajectories collected by a snapshot from somewhere in training. D4RL also includes some demonstration and goal-directed data. Importantly, the datasets are then tested on the same reward they were trained on (or the demonstrations they were targeting).

In contrast, our contribution is to use unsupervised exploratoy data collection to test offline RL algorithms. This creates datasets that facilitate multi-task offline RL from a single dataset and poses a very different challenge for offline RL than prior benchmarking work.

4.2.2 Unsupervised pre-training for RL

Recently, URLB [Laskin et al. 2021] proposed a benchmarking protocol for unsupervised pre-training algorithms in RL. In that work, each pre-training algorithm is allowed a fixed budget of unsupervised interactions with the environment and then oututs a pre-trained policy that is used to initialize an online RL algorithm at test time. In our work, instead of using a pre-trained policy as the output of the pre-training phase and then training online, we use the *dataset* as the output of the pre-training phase and then train entirely *offline*. To perform this data collection, we consider the same eight pre-training algorithms as in URLB [Pathak et al. 2017; Burda et al. 2019; Eysenbach et al. 2019; Pathak et al. 2019; Lee et al. 2019; Liu and Abbeel 2021b,a; Yarats et al. 2021a] as well as a uniformly random data collector.

There is also a line of purely theoretical work that studies the "reward-free RL" problem in the tabular and linear cases [Jin et al. 2020; Wang et al. 2020a]. Much like the goal of our protocol, these papers focus on collecting a dataset that allows for offline learning of a near optimal policy for any reward function. Our work can be viewed as an empirical evaluation of several different ways to scale up these reward-free RL algorithms to large domains with neural networks.

4.2.3 Multi-task reinforcement learning

Another important related line of work focuses on sharing data across tasks. Here, data collection is generally directed at multiple different goals with some combination of demonstrations and standard RL with task-specific rewards [Riedmiller et al. 2018; Cabi et al. 2019; Kalashnikov et al. 2021; Yu et al. 2021; Chen et al. 2021a]. In contrast, we focus on studying the capabilities of purely unsupervised data collection. We view these two research directions as complementary. For example, they could potentially be combined in future work, where unsupervised data could be added to multi-task data to facilitate transfer.

Another line of work generates goal-based multi-task data using different goals as the different tasks where the goals can be generated in an unsupervised manner [Andrychowicz et al. 2017; Ecoffet et al. 2019; Dendorfer et al. 2020; Mendonca et al. 2021]. Endrawis et al. [2021] in particular uses goal-based task generation to collect exploratory data to use with offline RL on robotic tasks. In contrast, we do not use any goal-based and focus our efforts on understanding how offline RL methods perform in the exploratory setting.

We should also note that our relabeling procedure is not novel and similar ideas have been used widely in the multi-task and goal-based RL literature [Andrychowicz et al. 2017; Riedmiller et al. 2018; Sekar et al. 2020; Eysenbach et al. 2020; Li et al. 2020; Kalashnikov et al. 2021].

4.2.4 Decoupling exploration and exploitation

A few recent position papers argue for separating the exploration and exploitation problems to facilitate scalable reinforcement learning [Levine 2021; Riedmiller et al. 2021]. Ideally, collecting large datasets in an unsupervised manner and re-using the data across many downstream tasks can dramatically improve data-efficiency on a per-task basis. Moreover, by splitting the problem into two, we can more effectively isolate the effects of each component of the algorithm. Our paper can be seen as taking a first step towards instantiating this vision of effectively decoupling exploration and exploitation.

4.2.5 Concurrent work

Concurrently to our paper, [Lambert et al. 2022] proposes a similar framework for unsupervised exploration followed by offline RL. In contrast to our work, they focus on proposing a novel MPC-based exploration technique and do not compare multiple offline RL algorithms in the exploratory setting, instead they only use CRR.

4.3 Our framework: ExORL

In this section we explicitly define our framework for exploratory data collection for offline RL (ExORL). The framework consists of three steps: collect, relabel, and learn. The rest of this section goes through each step in detail and the protocol is summarized in Algorithm 2.

```
Algorithm 2: ExORL: Collect-Relabel-Learn
Input: budget $B$, environment $\mathcal{E}$, data collection alg $C$, reward function $r$, offline RL alg $O$
Collect: Run $C$ on $\mathcal{E}$ to collect $B$ trajectories into a dataset $\mathcal{D}$ of $(s, a, s')$ tuples.
Relabel: Update each tuple in $\mathcal{D}$ to $(s, a, r(s, a), s')$. Call the relabeled dataset $\mathcal{D}_r$.
Learn: Run $O$ on the labeled data in $\mathcal{D}_r$.
```

We assume that we have access to a reward-free episodic MDP environment $\mathcal{E}$ with state space $\mathcal{S}$, action space $\mathcal{A}$, and stochastic transition dynamics determined by $P(\cdot|s, a)$. In our problem setup we are given an interaction budget B that limits the number of trajectories that we can collect in the collection phase. Rewards for relabeling can be defined by any function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

Collect. The first phase of ExORL is to collect the unlabeled exploratory data. To do this, we can use any algorithm C that at each episode k outputs a policy π_k conditioned on the history of prior interactions. Each episode collected by C is stored in the dataset $\mathcal{D}$ as a series of (s, a, s') tuples. We run the collection algorithm C until we have collected B episodes into the dataset $\mathcal{D}$.

In practice, we evaluate nine different unsupervised data collection algorithms as C in this phase. The first is a simple baseline that always outputs the uniformly random policy. The remaining eight are taken from URLB [Laskin et al. 2021] and can be sorted into three groups:

- Knowledge-based methods that maximize error of a predictive model: ICM [Pathak et al. 2017], Disagreement [Pathak et al. 2019], RND [Burda et al. 2019].
- Data-based algorithms that maximize some estimate of coverage of the state space: APT [Liu and Abbeel 2021b] and ProtoRL [Yarats et al. 2021a].

- Competence-based algorithms that learn a diverse set of skills: DIAYN [Eysenbach et al. 2019], SMM [Lee et al. 2019], and APS [Liu and Abbeel 2021a].

Relabel. Having collected a dataset $\mathcal{D}$ of (s, a, s') tuples, the next phase of our protocol relabels the data using the given reward function r. This simply requires evaluating $r(s, a)$ at each tuple in the dataset and then adding $(s, a, r(s, a), s')$ to the relabeled dataset $\mathcal{D}_r$.

In practice, we use either standard or hand-designed reward functions in each environment [Tassa et al. 2018; Laskin et al. 2021], and are detailed in Section C.7. We note, however, that our framework also allows for learning reward function (i.e. inverse RL or successor features).

Learn. Finally, we can learn a policy by running an offline RL algorithm $\mathcal{O}$ on our labeled dataset $\mathcal{D}_r$. The offline RL algorithm learns entirely offline by sampling tuples from the dataset and then outputs the final policy to be evaluated online in the environment $\mathcal{E}$ by calculating the return with respect to the reward function r.

In practice, we run five different offline RL algorithms for each reward function on each dataset. As a baseline we run simple behavior cloning. Then we run three different state of the art offline RL algorithms that each use a different mechanism to prevent extrapolation beyond the actions in the data. These algorithms are: CRR [Wang et al. 2020c] which uses filtered behavior cloning, CQL [Kumar et al. 2020] which uses pessimistic Q estimates, and TD3+BC [Fujimoto and Gu 2021] which regularizes toward the behavior. Finally, we run standard TD3 [Fujimoto et al. 2018b] as a baseline to test what happens when we run an off-policy RL algorithm that was originally designed for the online setting and has no mechanism explicitly designed to prevent extrapolation in the offline setting.

4.4 Experiments

In this section we conduct an empirical study to answer the following questions: **Q1**: Can the diversity of unsupervised data collected by ExORL enable vanilla off-policy RL agents to work in the offline setting? **Q2**: Is it possible to relabel this unsupervised data to facilitate multi-task offline RL? **Q3**: Is exploratory data necessary for multi-task offline RL? **Q4**: Would it be useful to mix exploratory data with task-specific data? **Q5**: How effective is ExORL as we scale the data collection budget?

4.4.1 Setup

We evaluate our framework on a set of challenging environments from the DeepMind control suite [Tassa et al. 2018]. Each environment has several different rewards that we use to evaluate multi-task generalization. See Section C.7 for detailed overview of the environments. We use URLB [Laskin et al. 2021] for unsupervised data collection with a budget of 1M reward-free environment interactions (1000 episodes) unless stated otherwise. The transitions in these datasets are then relabeled under a specific reward and used for offline RL. For offline RL we use five algorithms: BC, TD3+BC, CRR, CQL, and TD3. We adhere closely to the original hyper-parameter settings for each algorithm, but in several cases we perform hyper-parameter tuning to achieve best possible performance. See Section C.1 for more details. We train offline RL algorithms for 500k gradient updates and then evaluate by rolling out 10 episodes in the environment. We report mean and standard error across 10 random seeds. A description of computational requirements is provided in Section C.6.

4.4.2 Comparing offline RL algorithms on ExORL

Q1: Can the diversity unsupervised data collected by ExORL enable vanilla off-policy RL agents to work in the offline setting? **A1**: *Yes, vanilla off-policy RL agents can perform well from ExORL data, in particular vanilla TD3 outperforms the other offline RL algorithms..*

To verify this, we took three environments (Cartpole, Cheetah, and Jaco) and one reward for each environment (Swingup, Run, and Reach respectively). For each environment we run all nine exploration algorithm, but for clarity only present four in the main text that represent each of the main families of algrithms (ICM: knowledge-based, Proto: data-based, APS: competence-based, and Random as a baseline). We then evaluate with five different offline RL algorithms. Results are in Figure 4.3 and Section C.3.

Specifically, comparing the results on these datasets we see that vanilla TD3 consistently outperforms more sophisticated offline RL algorithms (TD3+BC, CRR, and CQL). This is a somewhat surprising result because the offline RL algorithms were designed specifically for the offline setting while TD3 was not. This emphasizes how the unsupervised data collected by ExORL provides a substantially different setting than the traditional offline RL problems that the algorithms were designed for. We hypothesize that this difference occurs because ExORL produces diverse datasets that allow for more aggressive trajectory stitching and less severe extrapolation than prior datasets.

While the relative strength of TD3 is consistent, the results are somewhat more varied across the different data collection algorithms. For example, in Jaco, the random data collection surprisingly outperforms all of the unsupervised exploration algorithms. This is due to over-exploration by the unsupervised algorithms which causes the arm to explore low-reward positions outside of the workspace, while the random agent keeps the hand closer to initialization. Also, while ExORL achieves performance comparable to or exceeding online RL in Cartpole Swingup and Jaco Reach, the performance on Cheetah Run is still not competitive with the optimal policy. This is likely due to insufficient exploration of the higher-dimensional state space in regions of high reward, suggesting there is still room to improve unsupervised exploration algorithms.

4.4.3 Multi-task learning with ExORL

Q2: Is it possible to relabel this unsupervised data to facilitate multi-task offline RL? **A2:** *Yes, multi-task offline RL can perform well from ExORL data.*

To confirm this we evaluate on one environment (Walker) across three different tasks (Stand, Walk, and Run). This means that each data collection algorithm only collects one dataset and we relabel it with three different rewards. Results are in Figure 4.4.

In particular, we are able to nearly solve both the Stand and Walk tasks from a single dataset collected with no knowledge of the rewards. Performance on Run is further away from optimal, but still strong enough to suggest meaningful progress towards the desired behavior. So, while there is room for improvement of the unsupervised exploration, the ExORL framework has clear promise to learn multiple tasks from a single task-agnostic dataset.

Again, we observe consistently strong performance of vanilla TD3 on exploratory data, confirming the result from the prior experiment. And comparing the different exploration algorithms, we see similar performance across each of the three representative algorithms (ICM, Proto, and APS) with each of them being much stronger than random.

4.4.4 Comparing ExORL to supervised data collection

Q3: Is exploratory data necessary for multi-task offline RL? **A3:** *Yes, the unsupervised exploratory data facilitates transfer where the supervised data does not.*

To demonstrate this, we consider different data-collection strategies ranging from unsupervised to supervised. *Unsupervised* corresponds to our standard ExORL approach with the Proto exploration algorithm. *Supervised* is similar to the approach in D4RL [Fu et al. 2020] and RL Unplugged [Gulcehre et al. 2020] of using the replay buffer of a policy trained for a task-specific reward. We use TD3 as the data collection agent and refer to the reward during training as the "data task". We then evaluate three offline RL agents (BC, TD3+BC, and TD3) on both the data task and a different "transfer task" that was not seen during training. Results in the Walker environment with Stand as the data task and Walk, Run, and Flip as the transfer tasks, see Figure 4.1(b).

For a second experiment we also add a *semi-supervised* strategy as a baseline. This strategy uses a TD3 agent to maximize the sum of the extrinsic reward from the data task as well as the task-agnostic intrinsic reward from Proto. We run this procedure on Cheetah and Walker using Run and Stand respectively as the data tasks and Run Backward and Walk as the transfer tasks, see results in Figure 4.5.

As we hypothesized, unsupervised data allows for strong performance on both data and transfers tasks while supervised data only facilitates strong performance on the data task while failing to transfer. Semi-supervised data was capable of facilitating transfer on the easier Stand-Walk pair, but failed more clearly on the Run-Run Backward task. This is likely because Stand is a pre-requisite to Walk, while successful Run and Run Backward behaviors cover much more different parts of the state space.

Examining the offline RL algorithms, we find that TD3+BC is the strongest on the supervised data, but struggles to transfer from unsupervised data. This makes sense because the TD3+BC algorithm was designed for the D4RL benchamrk that consists of supervised-style datasets. In contrast, vanilla TD3 performs stronger on the unsupervised data. Unsurprisingly, BC can work fairly well on supervised data, but fails on the unsupervised data and the transfer tasks.

Interestingly, vanilla TD3 outperforms TD3+BC on the semi-supervised data, suggesting that data diversity can facilitate standard TD learning without offline modifications. We will build on this insight in the next subsection.

4.4.5 Combining ExORL with supervised data

Q4: Would it be useful to mix exploratory data with task-specific data? **A4:** *Yes, it is useful to mix even a small amount of unsupervised data to facilitate transfer.*

To support this, we consider mixing datasets rather than modifying the data collection algorithms. We consider the same pairs of data and transfer tasks as in the prior subsection, but this time we mix the supervised and unsupervised datasets themselves. To do this, we randomly sample trajectories from the supervised and unsupervised datasets with varying proportions of unsupervised data while keeping the size of the resulting mixed dataset constant at 1M transitions (1000 episodes). We can then see the performance of BC, TD3+BC, and TD3 on datasets with varying proportions of unsupervised data in Figure 4.6.

Specifically, on the transfer tasks, TD3 achieves nearly the same performance with 25% unsupervised data as with 100%. Similarly, on the data task adding a relatively small amount of unsupervised data allows TD3 to match the performance of TD3+BC on the fully supervised data. This sends a very hopeful message that even adding a small amount of unsupervised exploratory data can facilitate both transfer to novel tasks and prevention of issues of extrapolation error for vanilla RL algorithms in the offline setting.

Comparing the offline RL algorithms more directly, we see that TD3 effectively takes advantage of the mixed datasets while TD3+BC and BC do not. By attempting to constrain to be near the behavior policy, algorithms like TD3+BC and BC can struggle to deal with data that is generated by a mixture of different behaviors, especially when these behaviors are not optimizing the test-time task.

4.4.6 Scaling of ExORL

Q5: How effective is ExORL as we scale the data collection budget? **A1:** *In general, all algorithms see improved performance with increased data collection budget.*

To confirm, this we collect datasets in the Walker domain with data collection budgets ranging from 100k to 10M transitions (100 to 10k episodes) using all nine unsupervised data collection algorithms. We then run offline RL on these datasets for two different rewards: Walk and Run. To compare the different datasets we run offline TD3 on each (since TD3 was the strongest performer in prior experiments). Then to compare the offline RL algorithms, we run each of our five offline RL algorithms on the datasets generated by ICM (since ICM was the strongest performing dataset with TD3). Results are in Figure 4.7.

First, we evaluate the results across the different exploration algorithms. All algorithms see returns to scale, but we do observe gains beginning to saturate by 10M transitions on the Run task before reaching optimal performance suggesting there is still room for improved algorithms. In low-data settings we see the best performance from DIAYN while in the high-data regime ICM begins to dominate. We note that the random agent also sees return to scale, suggesting that the Walk task is not a difficult exploration problem.

We can also compare across the different offline RL algorithms on the ICM data. Again we see clear returns to scale as performance improves with dataset size. As in previous experiments we see that TD3 is the strongest algorithm and this holds true across dataset sizes.

4.5 Discussion

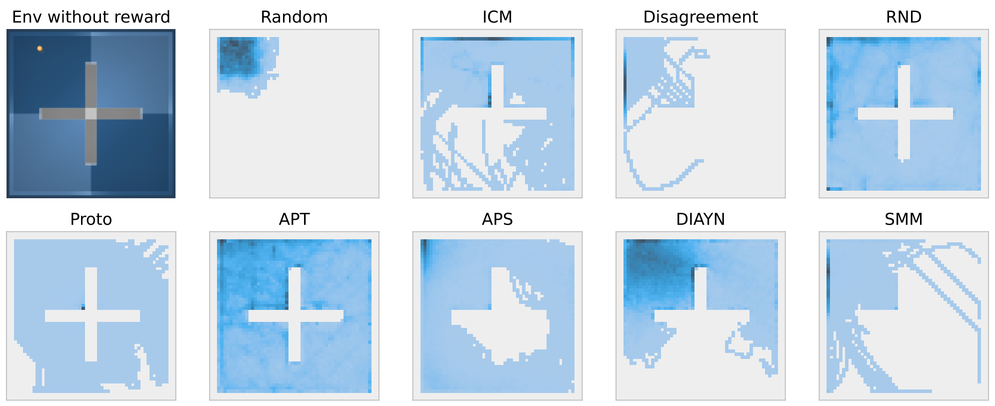
In this paper we propose ExORL – a novel framework for exploratory data collection for offline RL. Through an extensive empirical evaluation we conclude that using diverse data can greatly simplify offline RL algorithms by removing the need to fight the extrapolation problem. We then demonstrate that exploratory data is more suitable for efficient multi-task offline RL than standard task-directed offline RL data. Finally, we release ExORL's datasets as well as the experimentation code to encourage future research in this area. Now we will discuss a few of the possible directions for future research with ExORL.

Designing adaptive offline RL algorithms. Our dataset release provides a new testbed for improving offline RL algorithms. Our results suggest that previously designed offline RL algorithms perform well on supervised data, but are beaten by TD3 on unsupervised ExORL data. Ideally, an offline RL algorithm would be able to automatically adapt to the given dataset to recover the best of both worlds. There is a growing literature on offline hyperparameter tuning for offline RL that may be useful for approaching this challenge [Paine et al. 2020; Kumar et al. 2021b; Zhang and Jiang 2021; Fu et al. 2021a; Lee et al. 2021].

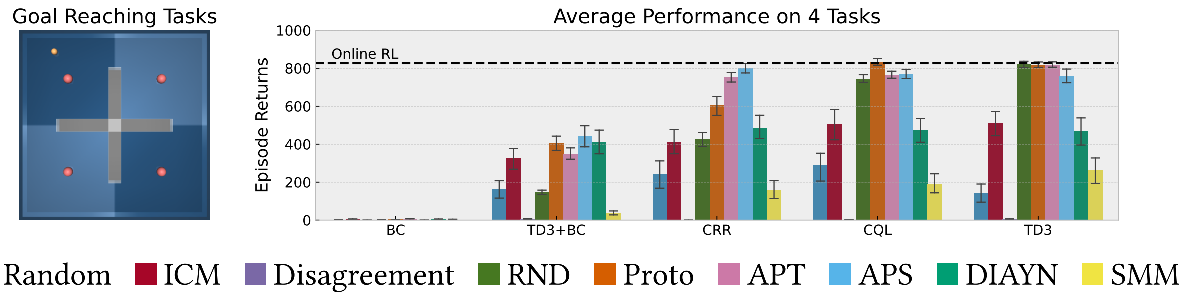
Improving unsupervised data collection. Our approach also yields a novel evaluation protocol for unsupervised RL by evaluating the generated dataset directly via offline RL. This differs from prior work on unsupervised RL [Laskin et al. 2021] by emphasizing the dataset rather than the pre-trained policy. This allows for comparisons across different unsupervised RL algorithms without having to fine-tune online. In future work we hope to explore how to optimize the unsupervised exploration algorithms to improve performance of the downstream offline RL. ExORL provides a useful starting point and framework for this problem.

Adding exploratory data to improve stability. Our preliminary experiments suggest that adding even a small amount of unsupervised data can allow vanilla TD3 to outperform other offline RL algorithms. Future work in this direction could take this insight further. For example, currently most offline RL algorithms are essentially incorporating some sort of regularization into the algorithm, but our results suggest it could be fruitful to instead “regularize” the data itself by mixing in some exploratory data. This is potentially a useful technique that is under-explored in the literature.

Scaling ExORL to more challenging environments. While we offer a large scale study on several domains from the DeepMind control suite, we do acknowledge that it would be fruitful to scale our framework to more complicated environments and settings. Specifically, we would be interested to extend these ideas into challenging exploration problems like manipulation, and higher-dimensional or even image-based observation spaces.



(a) Collect: an unsupervised RL agent interacts with the environment in a reward-free manner and stores the acquired transitions into a dataset $\mathcal{D}$ of (s, a, s') tuples. We visualize state distributions of the pointmass for each collected dataset $\mathcal{D}$.



(b) Relabel and **Learn**: Each tuple in $\mathcal{D}$ is relabeled by a task-specific reward function to yield a new dataset $\mathcal{D}_r$ of (s, a, r, s') tuples (there are 4 different rewards in this example, one for each goal). Finally, we run an offline RL algorithm on each dataset $\mathcal{D}_r$ in order to learn a policy to maximize the specified reward. Here, offline RL is performed by various state of the art offline RL algorithms as well as standard supervised learning (BC) and off-policy RL (TD3).

Figure 4.2: Our ExORL framework exploratory data collection for offline RL consists of three phases: *collect* **(a)**, *relabel* and *learn* **(b)**. Here we showcase our method on a pointmass maze environment to demonstrate that it is possible to collect exploratory data in a completely unsupervised manner that enables effective multi-task offline RL.

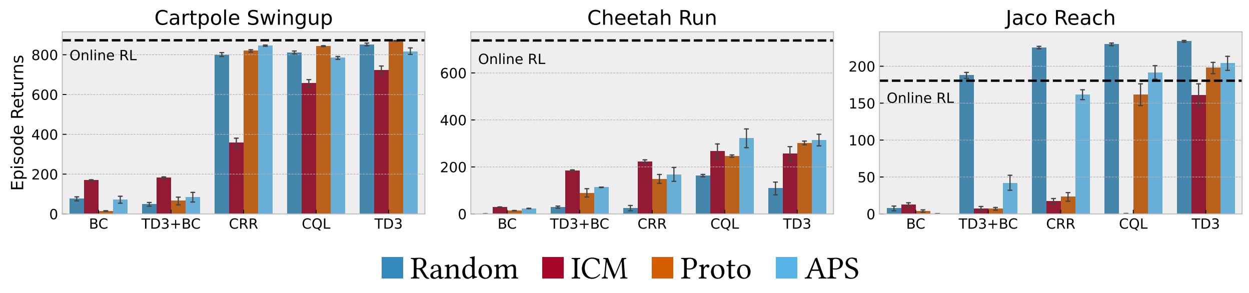


Figure 4.3: Offline evaluation of unsupervised datasets on one task for each of three different domains. Here we choose four representative unsupervised exploration algorithms, for full results see Section C.3. Vanilla TD3 usually outperforms all three offline RL algorithms.

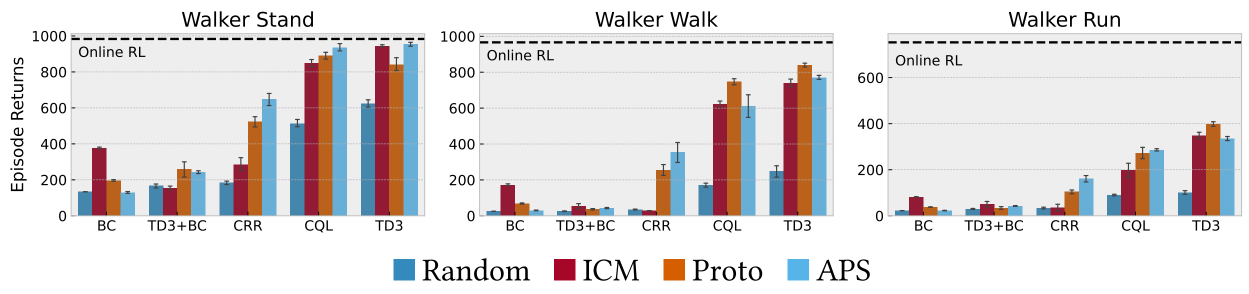


Figure 4.4: Offline evaluation of datasets for the Walker environment under three different rewards (Stand, Walk, and Run). We observe that ExORL allows for data relabeling to enable multi-task offline RL. See Section C.4 for full results across all exploration algorithms.

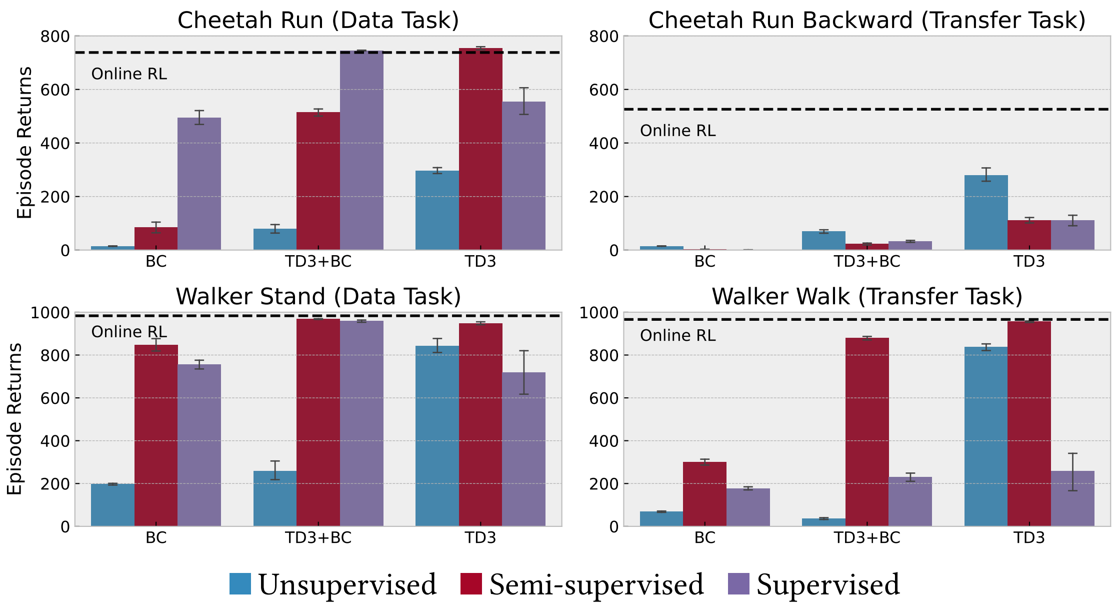


Figure 4.5: A comparison of three different data-collection strategies: unsupervised (intrinsic reward only), semi-supervised (intrinsic reward plus data task reward), and supervised (data task reward only). Data diversity is key to enable more aggressive trajectory stitching and task transfer.

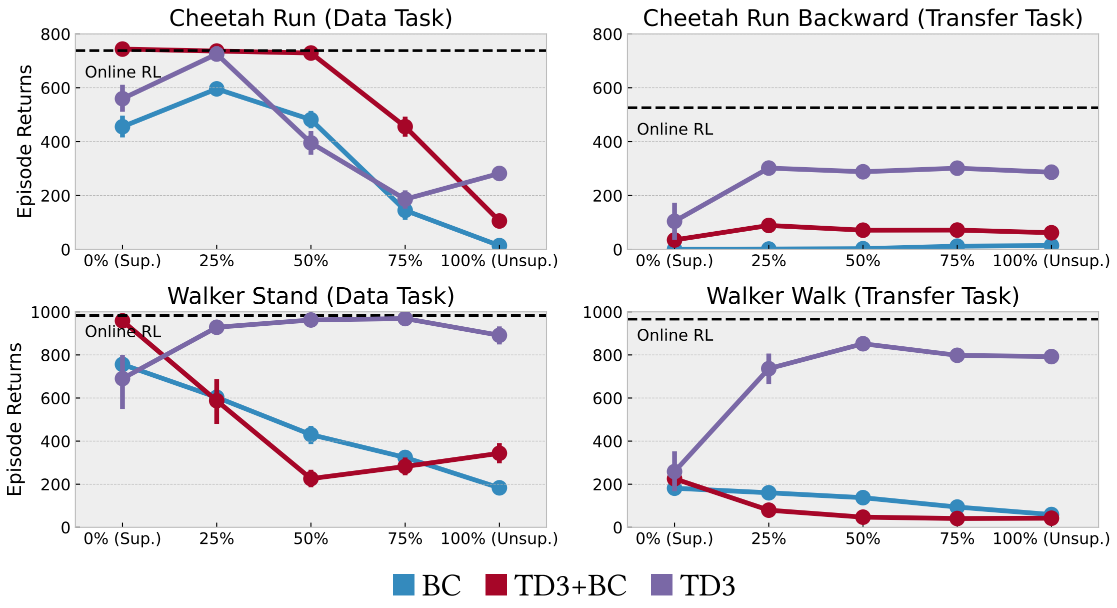
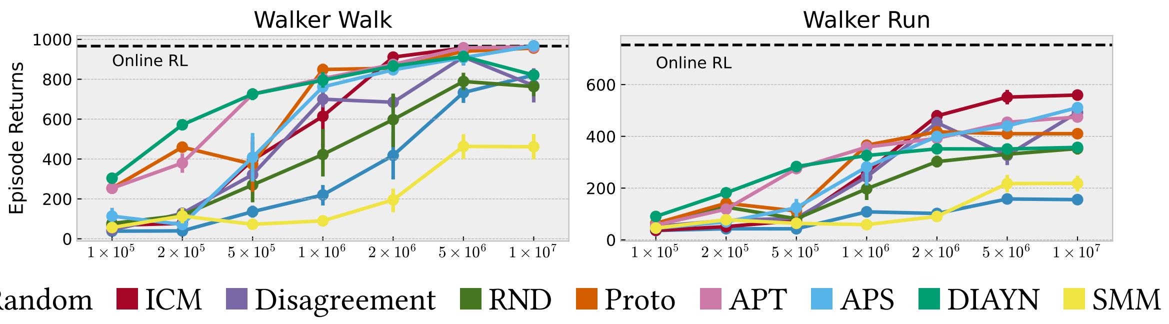
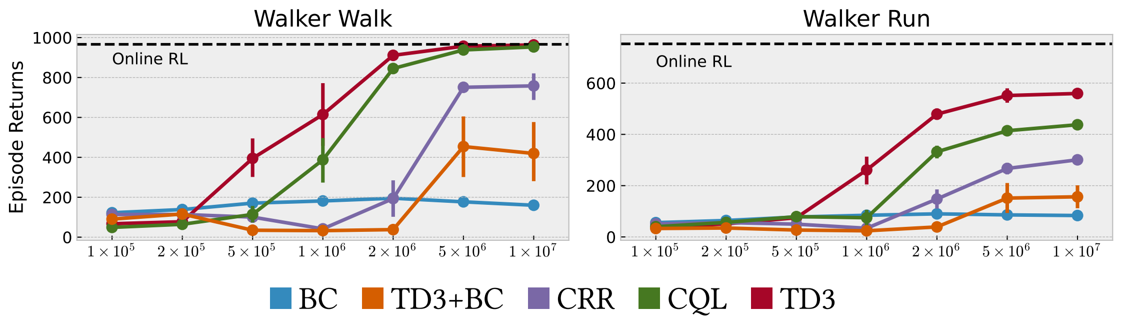


Figure 4.6: The effect of mixing supervised data (collected by TD3 on the data tasks) with unsupervised data (collected reward-free by Proto). Unsupervised data allows easier transfer and more efficient generalization to unseen tasks.



(a) Correlation between TD3 performance and the unsupervised data collection transition budget.



(b) Correlation between performance of various offline RL algorithms and unsupervised data collection transition budget for ICM.

Figure 4.7: Testing the impact of the size of unsupervised data collection budget on downstream offline RL performance. Figure 4.7(a) provides a breakdown by datasets, while Figure 4.7(b) provides a breakdown by offline RL algorithms. Generally, larger datasets demonstrate better performance across both exploration and offline RL algorithms.

5 | Visual Backtracking Teleoperation

5.1 Introduction

A common approach to control from images is to collect demonstrations of task success and train a behavioral cloning (BC) agent [Pomerleau 1988; Ahn et al. 2022; Jang et al. 2022]. This can lead to policies that are able to succeed on many tasks, particularly when mistakes do not cause the policy to move too far from the distribution of state-action transitions seen in the dataset of successful demonstrations. But, for many tasks (like T-shirt grasping) a mistake will take the policy out of this distribution and then the learned BC policy will fail to recover [Ross et al. 2011].

Such failures occur in part because a dataset of only successes does not contain enough information to recognize failures or learn recovery behaviors. To remedy this issue, we propose a novel data collection method called Visual Backtracking Teleoperation (VBT). Specifically, VBT leverages the teleoperator to collect visually similar failures, recoveries, and successes as seen in Fig. 5.1.

VBT data collection is designed to combine well with offline reinforcement learning (OffRL) rather than BC. VBT data contains the necessary coverage of the state action space to learn accurate value functions as well as recovery behaviors. Then OffRL can leverage sparse rewards to automatically emulate the advantageous actions (including recovery behaviors) while avoiding the sub-optimal actions that led to the initial task failure.

It is crucial to the VBT method that the failures, recoveries, and successes are visually similar.

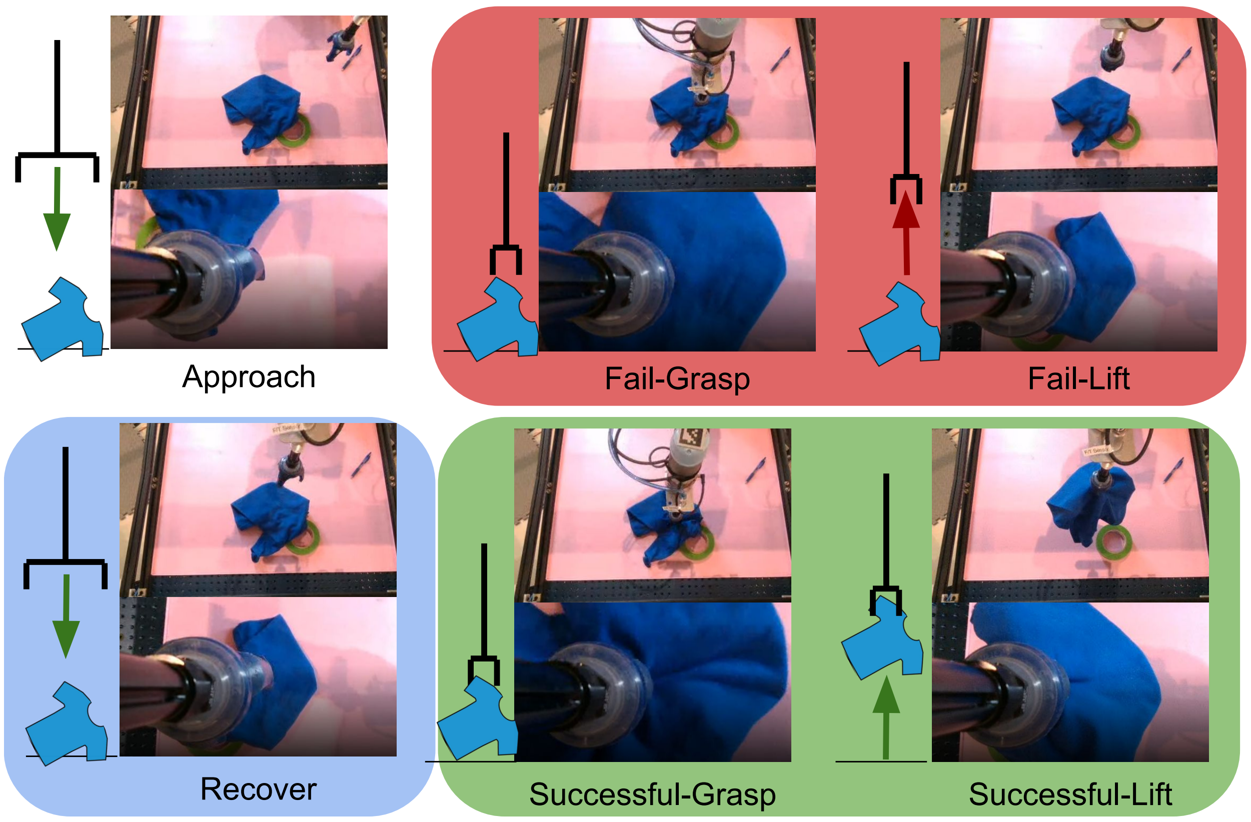


Figure 5.1: An illustration of our VBT method on a T-shirt grasping task. The method asks the teleoperator to demonstrate failure (red), recovery (blue), and success (green) within each trajectory. This provides the necessary coverage of failure and recovery to learn accurate value functions and robust policies while also preventing overfitting by ensuring that failures and successes are visually similar except for the task-relevant details.

Without this visual similarity, the OffRL learner can overfit to non-task-relevant elements of the image such as background clutter, leading to useless value functions. To easily maintain visual similarity, VBT collects each of failure, recovery, and success *within* the same trajectory (Fig. 5.1). Thus, VBT avoids overfitting, even on small, image-based datasets, by ensuring that differences between observations of failure and success are task-relevant.

Concretely, our contributions are:

1. We propose the novel VBT protocol for data collection to leverage human teleoperation to collect image-based datasets for use with OffRL. VBT resolves the two main issues with

naive methods: (a) lack of coverage of failures and recoveries and (b) overfitting caused by visually dissimilar failures and successes.

2. We discuss how and why VBT can enable better policy learning via learning more accurate Q functions and present empirical evidence of the improvement in Q functions learned from VBT data compared to several baselines for data collection.

3. We present real robot results on a deformable grasping task to demonstrate the effectiveness of VBT data. When training from scratch on just one hour of robot time for data collection and image-based observations, a policy trained with OffRL on VBT data succeeds 79% of the time while BC trained on successful demonstrations has a success rate of 66%.

5.2 Related Work

Our work falls into the broader category of learning from demonstrations [Schaal 1996]. But, rather than taking a more traditional approach of BC from demonstrations of success [Pomerleau 1988; Ahn et al. 2022; Jang et al. 2022; Argall et al. 2009], we propose a novel method for collecting demonstrations of failure and recovery as well as success. The inclusion of failures in our dataset leads us to use OffRL [Levine et al. 2020; Fujimoto et al. 2019c] to ensure that we do not imitate the failures. Specifically, we use the IQL [Kostrikov et al. 2021b] and AWAC [Nair et al. 2020] algorithms for OffRL. Some theoretical motivation for the use of OffRL rather than BC when learning from suboptimal data can be found in [Kumar et al. 2021a].

The insight that observations of failures is useful for policy learning has been made before in work on using success detectors to compute rewards in RL [Kalashnikov et al. 2021]. In contrast, VBT does not attempt to learn an explicit success detector, but provides a mechanism for collecting data that is especially useful for learning policies by capturing the salient differences between failure and success in visually similar observations.

While we consider a setting where training happens entirely offline (i.e., the data collection step is separated from the policy learning), there is a related line of work that collects examples of failures and recoveries by moving to an interactive setting where actions from partially trained policies are executed on the robot and judged by the human teleoperator. In particular the DAgger line of work exemplifies this pattern [Ross et al. 2011; Kelly et al. 2019; Mandlekar et al. 2020; Hoque et al. 2021, 2022]. In contrast, VBT operates completely offline and does not require the teleoperator to interact with the learning process or to deploy learned policies on the robot during training.

VBT also is particularly suited to efficiently learn image-based value functions for sparse reward tasks. This capability could be especially useful in architectures like SayCan [Ahn et al. 2022] that require image-based affordances for a variety of manipulation tasks, and it is an interesting direction for future work to use VBT as a component in training such larger systems.

5.3 Motivation

VBT is designed to solve two issues that arise from simpler data collection methods: (1) lack of coverage of failure and recovery behaviors, and (2) overfitting issues that arise in the low-data and high-dimensional observation setting. Here we describe both of these issues in more detail before explaining how VBT resolves them.

5.3.1 Lack of coverage of failure and recovery

Any offline learning algorithm will be fundamentally limited by the quality of the dataset. We can only expect the algorithm to reproduce the best behaviors in the dataset, not to reliably extrapolate beyond them. So, when collecting datasets for offline learning of value functions and policies, we want to ensure sufficient coverage of the relevant states and actions. We argue that for sparse reward robotics tasks this requires including failures and recoveries in the dataset.

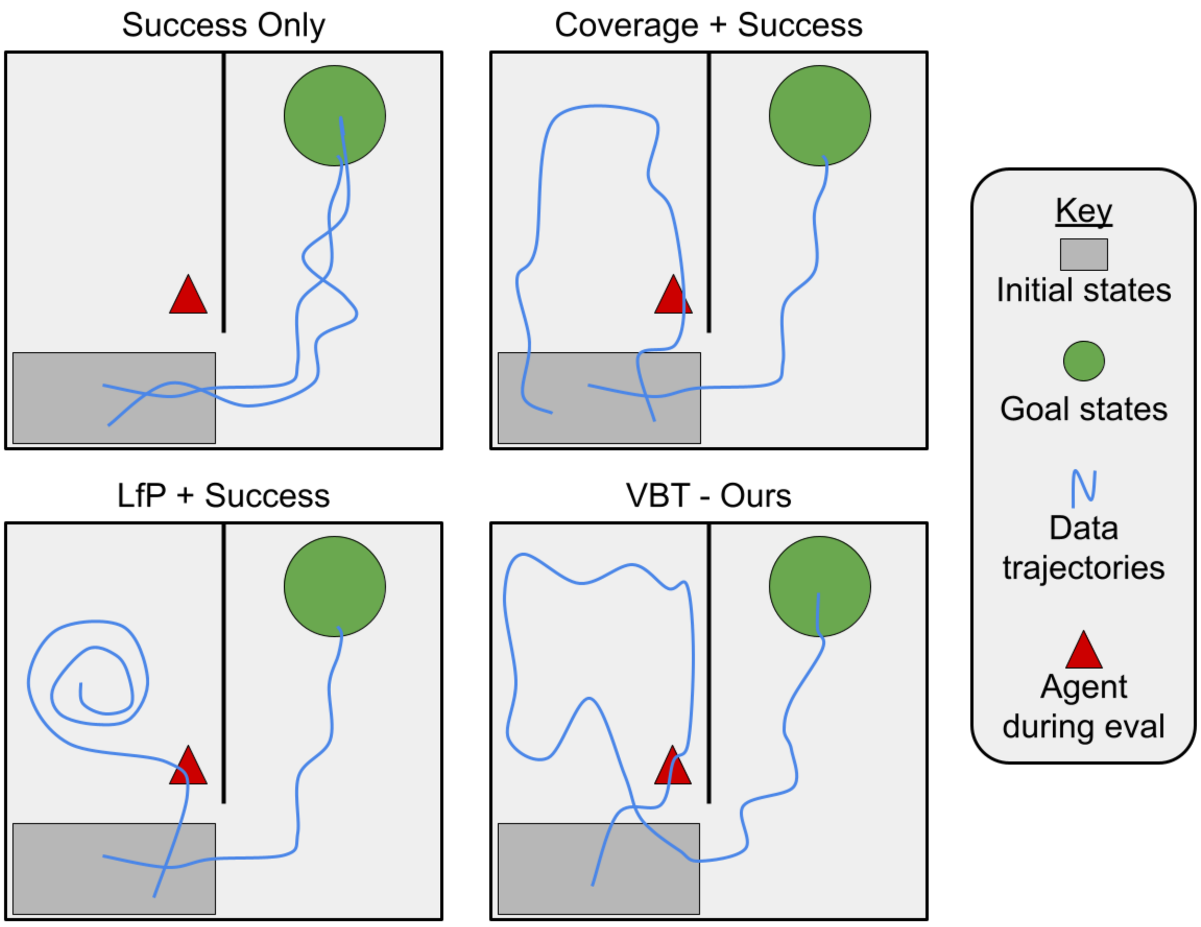


Figure 5.2: An illustration of each of the four types of datasets that we consider in a gridworld environment with a sparse reward for reaching the green goal. The Success Only dataset contains two successful trajectories. The Coverage+Success and LfP+Success datasets each contain one success and one failure. Our VBT dataset contains one trajectory that fails, then recovers, and then succeeds. Full descriptions of the datasets are in Section 5.5.

Value functions require coverage. Most OffRL algorithms involve estimating the Q and V value functions of a learned policy π that is different from the policy (in our case the teleoperator) that collected the dataset. Explicitly, letting $\gamma \in [0, 1)$ be a discount factor and r be the reward function they estimate $Q^\pi(s, a) = \mathbb{E}_\pi[\sum_{t=0}^\infty \gamma^t r(s_t, a_t)|s_0 = s, a_0 = a]$ and $V^\pi(s) = \mathbb{E}_{a\sim\pi|s}[Q^\pi(s, a)]$, where expectations are taken over actions sampled from π. For the rest of the paper we will omit the superscript π when clear from context.

The key issue with learning value functions for a policy π that did not collect the data is that we can only reliably estimate value functions at states and actions that are similar to those seen in the training set [Levine et al. 2020]. This is born out by theoretical work which often requires strong coverage assumptions to learn accurate value functions, such as assuming that the data distribution covers all reachable states and actions [Chen and Jiang 2019].

While this sort of assumption is too strong to satisfy in practice, we argue that for the sparse reward robotic tasks that we consider, the relevant notion of coverage is to include failures and recoveries, as well as successes, in the dataset. These failures and recoveries can provide coverage of the task-relevant states and behaviors necessary to learn useful value functions. Without failures the learned value functions will not be able to identify the "decision boundary" between failure and success that is necessary for reliably accomplishing the specified task. Much as in supervised learning it is difficult to train a binary classifier without any examples of the negative class, we conjecture that it is difficult to learn accurate Q functions for sparse reward tasks without any failures.

Consider the example datasets shown in Fig. 5.2. If we use the Success Only dataset to train a Q function and then query the Q function at the location of the red agent during evaluation we will get inaccurate Q values. The learned Q function does not have any indication that the presence of the wall between the agent and the goal exists since there is no training data from the left side of the wall. In contrast, any of the other datasets that contain failure examples where the agent enters the room to the left of the wall will allow the learned Q function to assign lower values to states left of the wall compared to those right of the wall.

Policies require coverage. Much as with value functions, learned policies should not be expected to produce behaviors that are not present in the training set. This can be seen by looking at the policy loss function used in OffRL algorithms like IQL [Kostrikov et al. 2021b] and AWAC [Nair et al. 2020] which take the following form (where $\mathcal{D}$ is the distribution that generates the dataset and Q, V are the learned Q and V value functions):

$$L(\pi) = \mathbb{E}_{s,a\sim\mathcal{D}}[\exp(Q(s,a) - V(s)) \cdot (-\log \pi(a|s))] \tag{5.1}$$

Notice that this is simply the standard negative log likelihood loss that is used for BC, except that each term is weighted by the exponentiated advantage function. This loss leads to policies that imitate actions with high advantages and ignore actions with low advantages. The key takeaway is that OffRL can only produce policies that choose actions that are already covered by the data distribution. So, if we want a robust policy that can recover from failures, we need to see recovery actions in the training set.

Note the DAgger line of work [Ross et al. 2011] raises a similar issue and resolves it by introducing online learning where the policy π is executed on the system to expand coverage. Instead, we are considering an entirely offline setting where we want to produce the necessary coverage of failure states and recovery behavior a priori from teleoperation.

To see an example of why coverage of failures and recoveries is necessary for robust policy learning, again consider the datasets shown in Fig. 5.2. Imagine that at evaluation time, the agent reaches the red triangle. If the agent was trained on the Success Only dataset, it must rely on extrapolation to select an action, but there are never any examples of "down" in the dataset. As a result, the agent is stuck in the room to the left of the wall and cannot recover. In contrast, the other datasets may allow for the agent to learn a useful recovery behavior since they have better coverage.

5.3.2 Overfitting in the low-data and image observation setting

When collecting data via teleoperation we are usually in the low-data regime since we are bottlenecked by teleoperator time on the real robot. Moreover, in the tasks that we consider here, the observation space is image-based and thus very high-dimensional. This combination of low-data and high-dimensional observations makes overfitting likely.

As explained above, in sparse reward tasks it is important for the Q functions and policies to accurately represent the "decision boundary" between failure and success in the task. In simple observation spaces like the gridworld in Fig. 5.2, this may be fairly easy, e.g., as in identifying a failure by recognizing that the (x, y) coordinated of the position are in the room to the left of the wall. However, in high-dimensional image-based observation spaces that are encountered in real robotic tasks, this can become much more challenging.

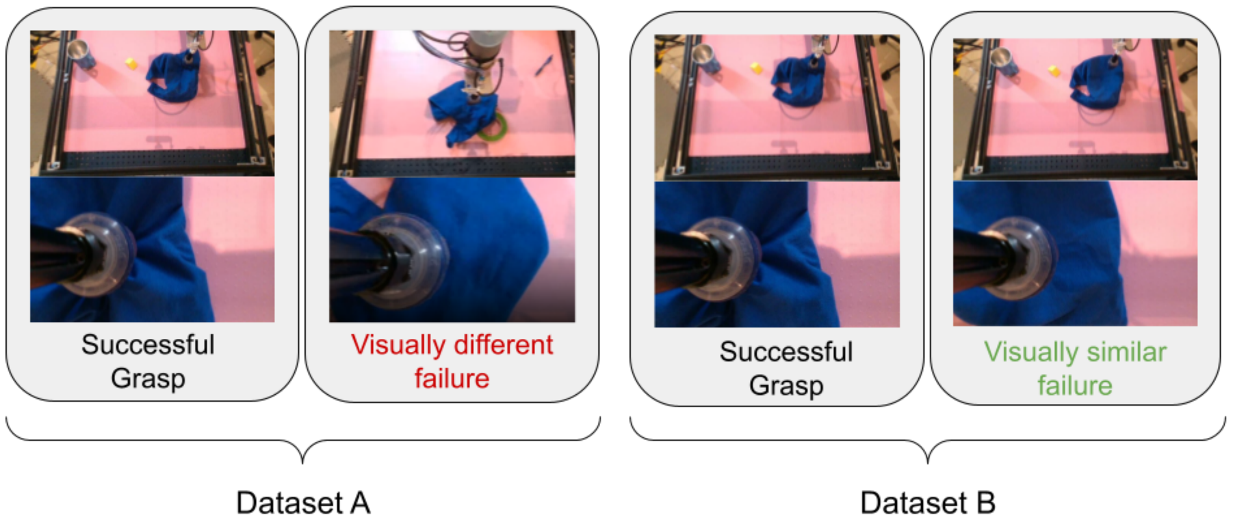


Figure 5.3: Datasets that include visually dissimilar successes and failures like Dataset A can cause overfitting. For example, a learned Q function could learn to attend background distractions (like the pen) instead of the task-relevant details. In contrast, in Dataset B the only differences between success and failure are task-relevant.

Fig. 5.3 illustrates the challenges of image observations in our grasping task. If successes and failures are collected naively in separate episodes (Dataset A), there will be many visual differences between success and failure beyond the salient details about the gripper and shirt. For example, background clutter like the presence of a pen or the exact configuration of the folds of the shirt can spuriously correlate with failure. In contrast, Dataset B reduces the chances of overfitting by eliminating the spurious correlations. Separating the examples in Dataset B requires learning a model that is attuned to the subtle task-relevant visual differences between failure and success that can generalize well. Our method aims to collect data like Dataset B.

5.4 Our Method: Visual Backtracking Teleoperation (VBT)

Our main contribution is a novel method for data collection called Visual Backtracking Teleoperation (VBT). This method is particularly suited to collecting small image based datasets. The main idea is that in order to learn useful Q functions and robust policies, we need a dataset that contains failures and recoveries as well as successes. Moreover, when learning from visual inputs we need the failures and recoveries to be as visually similar to the successes as possible to prevent overfitting and to encourage Q functions and policies to use only the task-relevant information in the images.

Explicitly, VBT data collection consists of three steps for the teleoperator *within* each trajectory:

1. *Failure:* the teleoperator first fails at the desired task.
2. *Recovery:* the teleoperator recovers from the failure and begins to attempt the task again.
3. *Success:* the teleoperator successfully finishes the task.

Then the data is labeled with a sparse reward signal: 1 for the final transition and a small penalty for all other transitions.

VBT is a general recipe that can be applied to any sparse-reward task that has well-defined failure modes and is amenable to teleoperation. The failure modes need not be unique and the data could consist of several different types of failures.

For our example task of deformable grasping these steps can be implemented as depicted in Fig. 5.1:

1. *Failure:* the teleoperator fails to grasp the shirt and begins to lift the arm.

2. *Recovery:* the teleoperator opens the gripper and moves the arm back towards the shirt.

3. *Success:* the teleoperator grasps and lifts the shirt.

Compared to a dataset that contains only demonstrations of success, VBT contains better coverage of failure and recovery behaviors. As explained in Section 5.3, this coverage is necessary to learn robust policies capable of recovery and Q functions that can recognize failure.

Compared to a dataset that naively mixes failures and successes that are collected separately, VBT has the benefit of visually similar failures and successes. As explained in Section 5.3, this prevents the learned Q functions and policies from overfitting based on visual details that are not task-relevant and instead forces the models to learn the task-relevant features that can generalize better.

5.5 Experimental Setup

5.5.1 Robot and Environment

Our setup consists of a reach-enabled [Wong et al. 2022] UR5 arm with a pneumatic powered 3-fingered gripper. We control the robot with a 5 dimensional action space: displacements in (x, y, z) no larger than 5cm, a gripper toggle to either an open or closed position, and a termination indicator that immediately terminates an episode. The orientation of the end-effector is fixed. If the agent does not select the terminate action within 100 steps, we automatically terminate the episode.

The workspace consists of the same blue T-shirt, with various distractor objects and variations in lighting. The robot observation space consists of 2 camera images (360x640x3) from a wrist camera and overhead camera, as well as the Cartesian position of the end-effector (x, y, z) and an indicator of whether the gripper is currently open or closed. We stack a sequence of 4 consecutive timesteps of observations as the input for our learning algorithms.

The task is to lift the T-shirt more than 80% off a table. During training, as explained in Section 5.4, we automatically label each successful trajectory collected by the teleoperator with a terminal reward of 1. We give all other transitions a reward of -0.01 to encourage faster completion of the task. This environment is illustrated in Fig. 5.4.

5.5.2 Datasets

To understand the performance of VBT, we collect several different baseline datasets of robotic T-shirt grasping by human teleoperators. Each dataset contains 60 minutes of data from the real robot which equates to approximately $10,000$ steps in the environment.

Success demonstrations (Success): The success dataset consists of demonstrations of task success. These episodes demonstrate efficient successful executions of the task with minimal recovery behaviors collected by a human teleoperator instructed to complete the task successfully. The episodes always terminate in success.

Visual Backtracking Teleoperation (VBT-Ours): Each episode in this dataset demonstrates failure, then recovery, and then task success as described in Section 5.4. The visuals of the background, lighting, and positions of scene objects are shared with both optimal behavior, bad

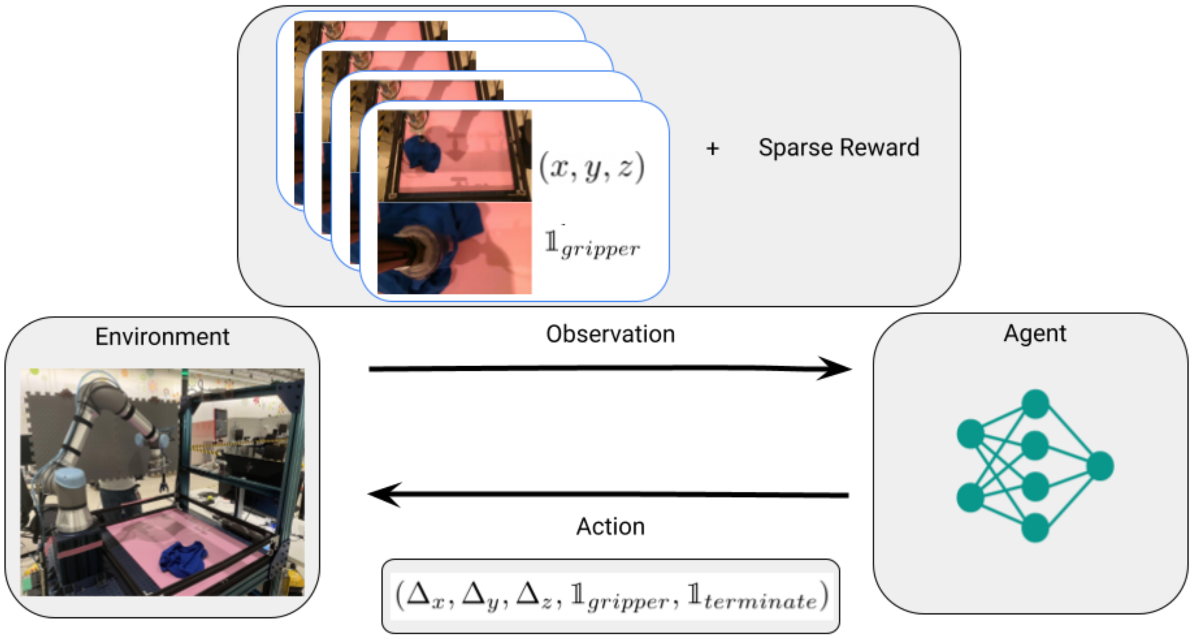


Figure 5.4: A description of the observation and actions spaces we use for our real robot experiments. The agent receives an observation of 4 timesteps of images, cartesian coordinates of the end effector, gripper status, and reward from the robot and sends back a 5 dimensional action of cartesian displacements and gripper and terminate commands.

behavior, and recoveries. The episodes always terminate in success.

Success mixed with failures and recoveries (Coverage+Success): This dataset is a mixture of two datasets containing 30 minutes of robot data from each one. The first is the Success dataset described above. The second is a dataset containing repeated failures and recoveries and intended to provide coverage of task-relevant behaviors. This dataset consists of many attempted grasps and retries in quick succession as well as many examples where the shirt is grasped, released, and then attempted to be re-grasped. The goal of this dataset is to provide coverage of failures and recoveries.

Success mixed with learning from play (LfP+Success): This dataset is also a mixture of two datasets containing 30 minutes of robot data from each one. The first is again the Success dataset. The second is a learning from play [Lynch et al. 2020] dataset which contains data of the robot demonstrating rich interactions such as rolling lifting, dragging, and pushing, between the robot arm and a variety of different objects beyond the T-shirt.

5.5.3 Learning algorithms

We run several different policy learning algorithms to understand utility of our datasets. **Behaviour Cloning (BC)** where policy is trained directly to imitate the actions in the dataset. **AWAC** [Nair et al. 2020] where an advantaged weighted actor-critic method is used to train both a policy π and critic able to evaluate V^{π} and Q^{π}. **IQL** [Kostrikov et al. 2021b] where implicit Q-Learning is used to train both a policy π and critics V^{π} and Q^{π}.

All policies and value functions are parameterized by the same neural architecture and trained using `JAX` [Bradbury et al. 2018] and `Flax` [Heek et al. 2020]. The image inputs are passed through a ResNet [He et al. 2016] encoder and then the features are concatenated along with the Cartesian coordinates of the gripper and the gripper indicator (as well as the action for Q functions). These features are then passed through a simple multi-layer perceptron to output either an action or value.

All policies and value functions are trained from 700K gradient steps using the Adam [Kingma and Ba 2014] optimizer. During training images are aggressively augmented using changes in brightness, sharpness, color, contrast, artificial shadows, rotation, and cropping. This allows us to train the encoders from scratch on relatively small datasets. We use an inverse temperature of 0.1 for both IQL and AWAC, and we use expectile 0.9 for IQL, following [Kostrikov et al. 2021b].

5.6 Experimental Results

We now present experimental results on our deformable grasping task that attempt to answer the following questions:

1. Are Q functions trained on VBT data more accurate than those trained on other datasets? And if so, why?
2. Are policies trained on VBT data more successful than those trained on other datasets? And if so, why?

5.6.1 VBT data leads to improved value functions

To see the improvement in Q functions when training on VBT data we present examples of the learned Q functions on representative trajectories. From these examples we show that training Q functions on VBT data resolves each of the issues raised in Section 5.3. Namely, the Q functions trained on VBT data (1) correctly identify failures while those trained on Success data do not and (2) resolve the overfitting issues that can arise when using Coverage+Success data.

First, we will show how Q functions trained on VBT data correctly identify failures and recoveries. To do this we illustrate the Q functions trained on Success, Coverage+Success, LfP+Success and VBT data on a single trajectory that contains failure, recovery, and success in Fig. 5.5. As explained in Section 5.3, the Success dataset does not provide sufficient information to understand failures. As a result, the Q function trained on Success does not use the image observation to recognize failure. Instead it uses the proprioceptive state to correlates "gripper closed and moving up" with higher values *regardless of whether the shirt is in hand*. Training on VBT data resolves this issue by forcing the Q function to attend to the information in the image to recognize when a failure has occurred. As a result, the Q function trained on VBT drops dramatically precisely when the missed grasp happens.

To ensure that these results generalize beyond this single trajectory, we compute averages of the relevant value functions across a held-out test of VBT-style trajectories. The results are shown in Fig. 5.6. Each trajectory contains three gripper toggles: is a missed grasp, an open gripper recovery behavior, and a successful grasp. The key observations are that as in Fig. 5.5, the value functions trained on VBT correctly identify that the Q value of the missed grasp is substantially lower than the V value of the state from which the grasp occurs. This means that the OffRL algorithms using these value functions will correctly learn to avoid missed grasps. In contrast, training on any of the baseline datasets leads to Q values that are above the V value for the missed grasp (meaning they will learn to miss). And similarly, training IQL on the baseline datasets yields Q values for the recovery behavior that are substantially below the V values while training on VBT does not.

The above results indicate that the Coverage+Success does not resolve the coverage issues that are preventing us from learning accurate Q functions. This may be somewhat surprising since this dataset was designed to have good coverage. We claim that this issue is due to overfitting in our low-data and high-dimensional observation setting. Essentially, the Q function trained on Coverage+Success data uses the image observation to classify the trajectory as either "success" or "failure". If the trajectory is classified as "success" then the Q function behaves just like a Q function trained only on Success and associates "gripper closed and moving up" with higher Q values. However, if the trajectory is classified as "failure" then the Q function trained on Coverage+Success data behaves differently and assigns uniformly lower Q values to the trajectory. Some hint of this can be seen in Fig. 5.6 where the values learned on Coverage+Success have substantially higher variance caused by some of the trajectories being classified as failures (note that LfP+Success behaves much more like Success alone since the classification problem between the two datasets is easy enough that the LfP data is completely ignored).

To isolate this issue, we plot histograms of the learned Q values on the train datasets and held out test datasets for both the Coverage+Success and VBT data in Fig. 5.7. The histograms confirm that the Q function trained on Coverage+Success assigns low values to Coverage transitions from the training set and high values to Success transitions from the training set. But, these Q functions do not generalize. The test set yields a very different distribution of values. In contrast, VBT data

Dataset	Policy	Task Success
Success	BC	66 ± 4%
Success	AWAC	67 ± 3%
Success	IQL	69 ± 3%
Coverage+Success	AWAC	52 ± 4%
Coverage+Success	IQL	64 ± 3%
LfP+Success	AWAC	62 ± 4%
LfP+Success	IQL	58 ± 4%
VBT-Ours	BC	73 ± 3%
VBT-Ours	AWAC	73 ± 3%
VBT-Ours	**IQL**	**79 ± 3%**

Table 5.1: Evaluation of grasping on an AB test of 1437 total episodes. Error bars report standard error.

resolves the overfitting issue by ensuring that the only visual differences between failure and success are task-relevant, which facilitates better generalization.

5.6.2 VBT data leads to improved policies

To measure the impact of switching to VBT data on policy performance we perform an AB test. Specifically, we train policies using each of the three learning algorithms (BC, AWAC, IQL) on each of the four types of datasets (Success, Coverage+Success, LfP+Success, VBT-Ours), but excluding BC on the datasets that contain episodes that do not terminate in success (Coverage+Success, LfP+Success). For the AB test, each episode a policy is chosen at random and executed until either the policy sends the "terminate" action or the maximum number of steps per episode is reached. Success is determined by the majority vote out of three human labelers based on the final image in the episode. The randomness of the AB test ensures a valid comparison between all of the trained policies. Results are reported in Table 5.1.

There are several takeaways from these results:

1. Training IQL on VBT outperforms standard BC on Success data by 13% and outperforms IQL

on Success data by 10%. These are significant gains, especially given that we are training each policy from scratch on very small datasets with image-based observations.

2. Due to overfitting, OffRL algorithms trained on Coverage+Success and LfP+Success underperform those same algorithms trained on Success data alone.

3. Even BC on VBT data outperforms BC on Successes for this task since training BC on VBT leads to retrying behaviors. OffRL provides further benefits beyond BC by effectively filtering out the suboptimal actions that are present in VBT data.

Overall, the experimental results paint a picture that VBT is able to improve coverage and prevent overfitting. This allows for more effective OffRL in terms of qualitatively better value functions and quantitatively better policies than we get from simple Success data or naively mixing in failures as in the Coverage+Success or LfP+Success data.

5.7 Discussion

In this paper, we introduced VBT, a protocol for robot data-collection for offline policy learning. We showed how value functions learned using VBT data tend to be more accurate, leading to robust policy learning with OffRL. We compared VBT to a number of other data collection protocols, and found that policies learned using VBT result in superior performance. We demonstrated these results on a real world, vision-based deformable manipulation task.

While our method significantly outperforms other data-collection protocols, it has certain limitations. VBT requires a teleoperator to understand the relevant failure modes for the task (in our grasping task, this corresponds to narrowly missing the T-shirt), and the ability to recover and re-attempt.

In future work, it would be interesting to apply VBT to tasks with more variety, such as collecting a dataset consisting of interactions with many different objects, instead of just one object. Finally, while we studied a relatively short-horizon task, future work can look into applying VBT to long-horizon tasks.

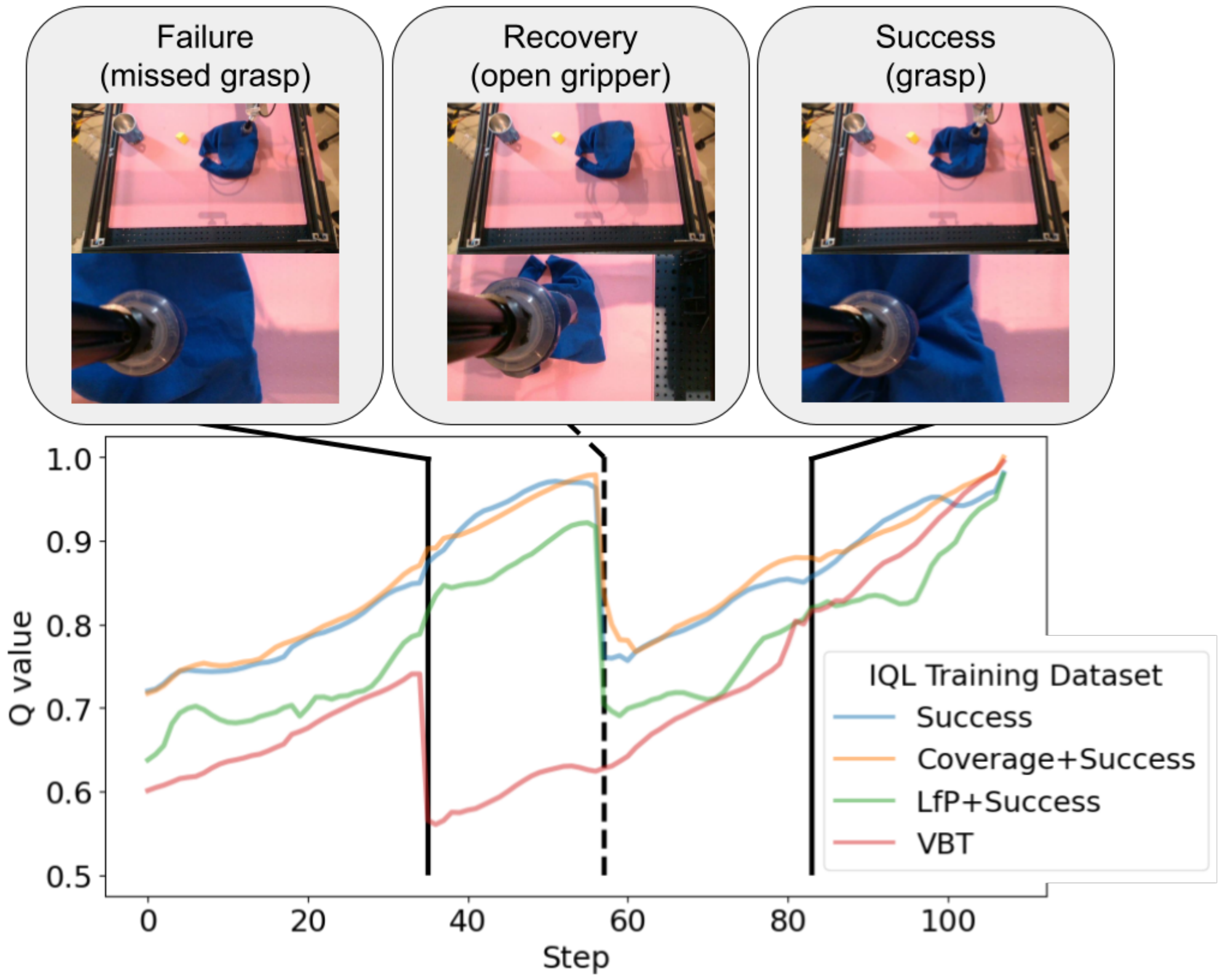


Figure 5.5: Evaluation of various Q functions on a single held out evaluation trajectory that is not seen during training. The trajectory consists of a failure followed by recovery and a successful grasp. The Q functions are trained by running IQL on different datasets (Success, Coverage+Success, LfP+Success or VBT). Notice that the VBT-trained Q function is the only one to drop precisely at the point of the missed grasp instead of the open gripper action that is necessary for recovery.

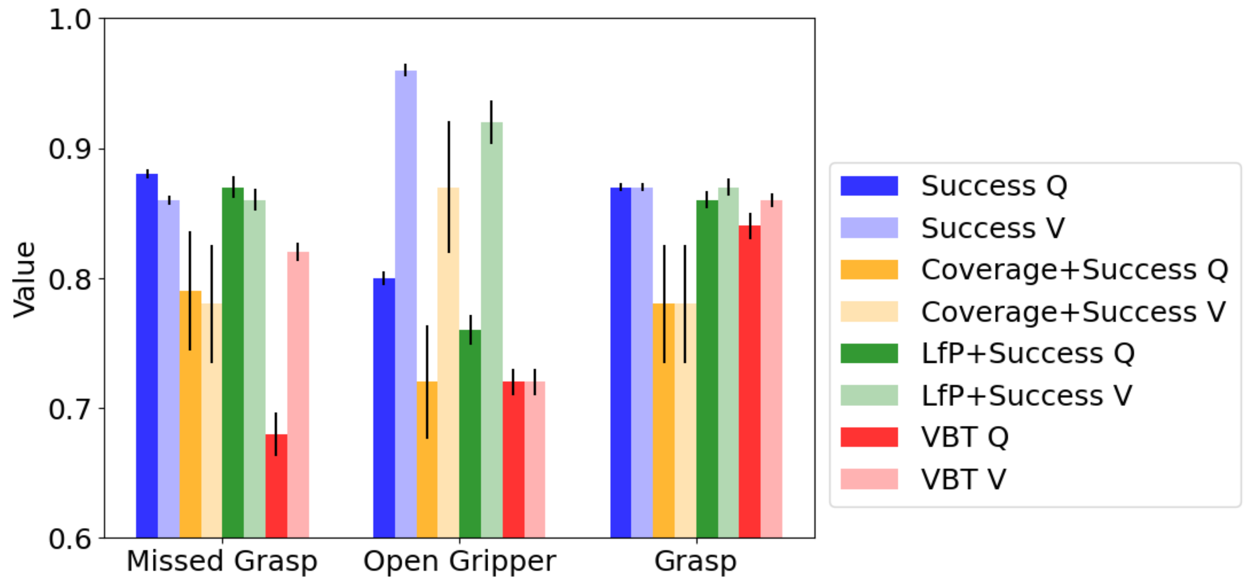


Figure 5.6: Visualization of the average Q and V values at each of the three key steps highlighted in Fig. 5.5 across a held-out test set of 35 trajectories. Error bars show standard error. Note that VBT has much lower Q than V for the missed grasp, while the baseline datasets have much lower Q than V for the open gripper, meaning the VBT value functions are more accurate.

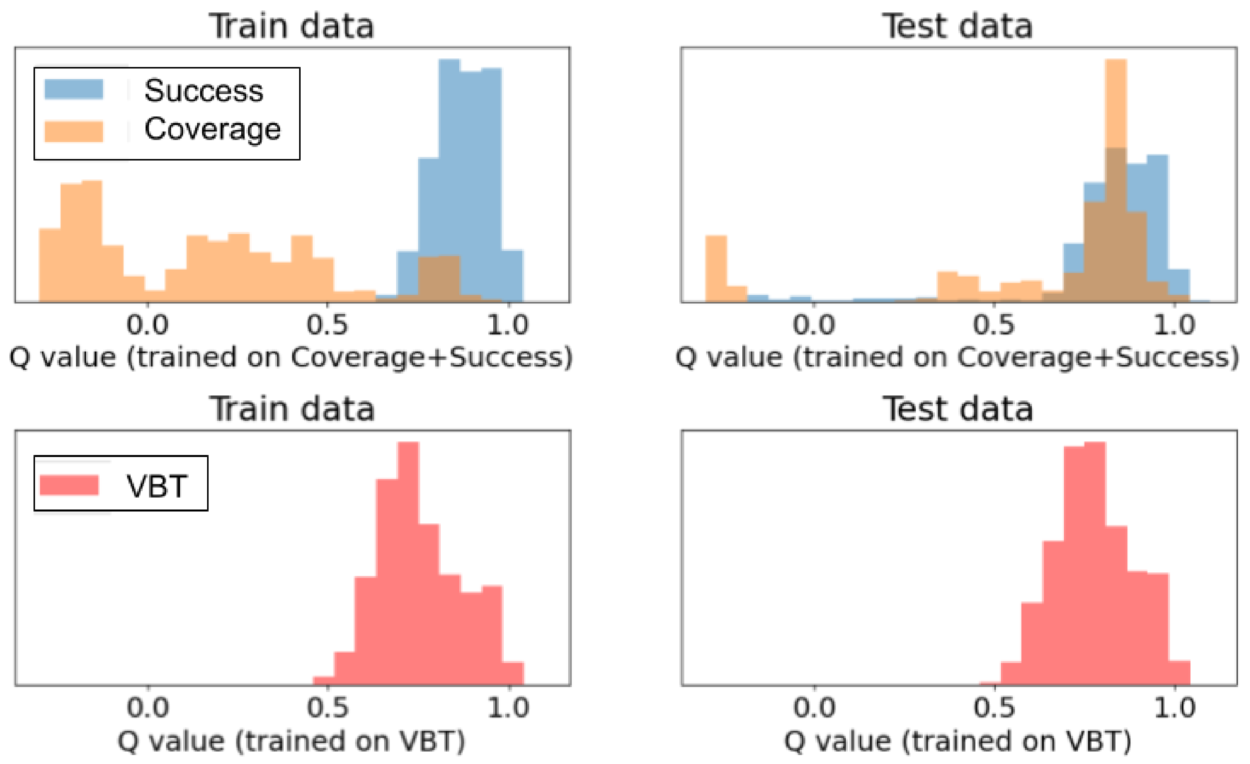


Figure 5.7: Histograms of the Q values of IQL trained on Coverage+Success (Top) and VBT (Bottom) on both the data that the respective Q functions were trained on (Left) and held out test sets from the same distribution (Right). The Q function trained on Coverage+Success demonstrates substantial overfitting in the mismatch between the distribution of Q values between train and test while the one trained on VBT does not.

Part III

Pretraining

6 Inverse dynamics pretraining for multitask imitation

6.1 Introduction

Pipelines in image recognition and natural language processing commonly use large datasets to pretrain representations that are then transferred to downstream tasks where data is limited [Devlin et al. 2018; Chen et al. 2020a; Radford et al. 2021a]. In this paper, we consider how this paradigm can be applied to imitation learning [Pomerleau 1991; Ho and Ermon 2016; Kostrikov et al. 2019]. In contrast to supervised learning domains where datasets consist of input-output pairs, imitation learning datasets consist of *trajectories* with both the input-output mapping to be learned (namely, observation-action pairs) as well as information about the dynamics of the environment. Given this additional structure, it is worthwhile to study pretraining approaches that can incorporate this structure to improve beyond methods from traditional supervised learning domains.

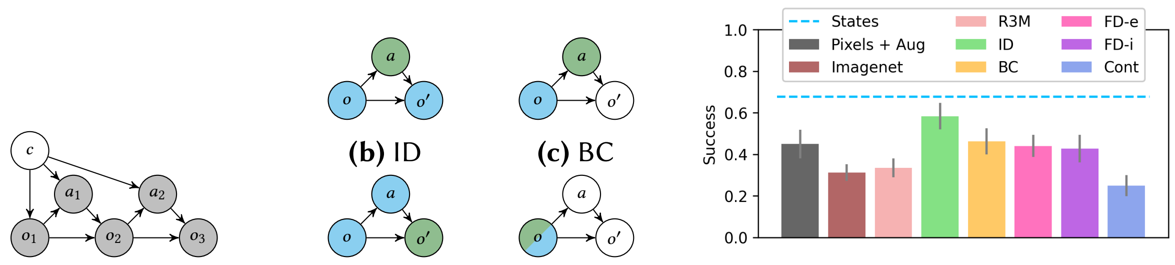
To formalize the precise notion of transfer between pretraining and finetuning phases, we consider a multitask imitation setting where the environment (i.e., the transition dynamics) is fixed and data is comprised of trajectories of *task experts* acting in this environment. A task is defined by a latent context variable that is observed by an expert demonstrator, but is not contained in the dataset, as shown in Figure 6.1. During pretraining, we have access to a large number of trajectories from various tasks, while during finetuning we have access to a small number of trajectories from a single task. The goal is thus to use the pretraining dataset to learn representations that contain information about the environment that facilitates efficient learning of the finetuning task.

A number of existing works have proposed objectives for representation learning that are applicable in this setting [Schwarzer et al. 2021; Stooke et al. 2021; Yang and Nachum 2021; Yang et al. 2023], and we consider a variety of algorithms and modes of analysis to determine which approach is the most promising. Algorithmically, we consider four generic classes of objectives for pretraining: inverse dynamics, behavior cloning, forward dynamics, and static observation modeling (Figure 6.1). We conduct two types of analysis. First, we conduct an extensive empirical evaluation and introspection of the candidate algorithms along with several strong baselines. Second, we present a simple but general theoretical model of the multitask representation learning problem and analyze the relative merits of the candidate algorithms under this model.

Our main results from these analyses are summarized as follows:

1. Across a broad array of experiments from visual observations in six environments, out of all approaches considered, inverse dynamics is the only one that consistently outperforms the baseline of training a model from scratch. The performance of inverse dynamics even matches that of finetuning from ground truth low-dimensional states on in-distribution contexts. Moreover, we find that inverse dynamics scales the best with pretraining dataset size and most effectively maintains relevant information about the observation in its learned representation.

2. In our simplified model of representation learning, we show that inverse dynamics pretraining efficiently recovers the ideal representation while behavior cloning can suffer from confounding and forward dynamics can suffer from poor sample efficiency. These results provide intuition for the empirical results and motivate why inverse dynamics pretraining is so performant and robust.



(a) Multitask imitation. **(b)** ID **(c)** BC **(d)** FD **(e)** Cont **(f)** Experimental results

Figure 6.1: (a) A graphical models of the setting. Shaded nodes indicate observed variables. The expert behavior (i.e., $o_i \longrightarrow a_i$) is determined by an unobserved context variable c while the transition dynamics (i.e., $(o_i, a_i) \longrightarrow o_{i+1}$) are determined by the environment dynamics. **(b)-(e)** illustrate the candidate algorithms. We use blue to indicate inputs to the algorithm and green to indicate prediction targets. ID = inverse dynamics, BC = behavior cloning, FD = forward dynamics, Cont = contrastive learning. **(f)** Shows success of policies finetuned on top of various representations averaged across all datasets in our suite for default dataset sizes. Inverse dynamics (shown in green) is the only method to substantially outperform the baseline of training from scratch (shown in black). Further details about the experimental protocol and results are in Sections 6.4 and 6.5.

6.2 Related work

As explained above, pretraining a representation has become a dominant paradigm in computer vision and natural language processing [Devlin et al. 2018; Chen et al. 2020a; Radford et al. 2021a]. Determining how to best leverage similar pretraining techniques in decision making problems is an important step towards extending the success of supervised learning into more temporally extended problems like those in robotics [Yang et al. 2023].

Prior work proposes several possible pretraining objectives for learning features for decision-making (and illustrated in Figure 6.1). First, inverse dynamics modeling has been proposed in several settings, although never as a representation learning algorithm for multitask imitation. Most directly related to our work is Efroni et al. [2021]; Lamb et al. [2022] which use multi-step inverse dynamics for feature extraction for exploration in reinforcement learning (RL) in the presence of exogenous noise. Later work from Islam et al. [2022] extended this approach to offline RL. Less closely related are Pathak et al. [2017] which uses inverse dynamics in the context of exploration and Baker et al. [2022]; Venuto et al. [2022] which use an inverse dynamics model to label video data with actions for imitation.

Another, perhaps simpler approach is to use behavior cloning as a pretraining algorithm. Arora et al. [2020] shows that this can be a well-motivated approach to pretraining a representation when the task variable is observed. Other work uses behavior cloning objectives to pretrain representations of temproally extended actions [Ajay et al. 2020] or priors for offline RL [Zang et al. 2022].

A third approach is to model the forward dynamics of the system as a pretraining objective. Most directly related to our work, Nachum and Yang [2021] show that this is a well-motivated technique for imitation learning and provide empirical evidence on single task atari games, but do not compare to inverse dynamics. This technique has also been explored in empirical work for online and offline RL [Schwarzer et al. 2021; Laskin et al. 2020; Aytar et al. 2018; Lee et al. 2022; Wu et al. 2023].

Finally, a method which we will refer to as static observation modeling does not leverage information about dynamics and rather directly uses self-supervised methods from computer vision [Pari et al. 2021; Chen et al. 2020a; Grill et al. 2020]. This approach does not take advantage of any additional structure in an imitation learning setting, but has nevertheless worked well in some settings.

Several empirical studies of representation learning for decision-making already exist. Most closely related to this work, [Chen et al. 2022] conducts an empirical evaluation of representations for imitation and finds that none of them consistently outperform training directly from pixels. However, this prior work (a) considers much larger finetuning datasets which can dramatically reduce the benefits of pretraining, and (b) considers different environments than we do, where the gap between pretraining and finetuning tasks is less controlled. Another line of work like Nair et al. [2022] attempts to pretrain general representations using large human-collected video datasets like Ego4d [Grauman et al. 2022]. In contrast, we focus on a more carefully controlled (albeit smaller scale) experimental settings where we can derive a more clear understanding of the relative merits of different pretraining objectives. Another empirical study from Stooke et al. [2021] considers representations in online reinforcement learning. Meanwhile, Yang and Nachum [2021] considers representations for imitation but does not consider image-based or multitask problems. Moreover, none of these works includes a theoretical understanding for the findings presented therein.

A further discussion of pretraining in the context of imitation can be found in Appendix D.1.

6.3 Problem setup

Here we present the formal setup for our problem setting of reward free pretraining from multitask expert data . We formalize this as a contextual MDP with rich (i.e., visual) observations where the latent context determines the initial state and reward functions.

Environment. We model the environment as a contextual MDP with context-independent dynamics:

$$c \sim P_c, \qquad o_0 \sim \rho_c, \qquad r_i = r_c(s_i, a_i), \qquad o_{i+1} \sim T(o_i, a_i). \tag{6.1}$$

Importantly, we consider the context variable c and rewards r_c to be *latent*, i.e., they are not available during training, and only used to evaluate a learned policy. At a high level, this captures the setting where the task (defined by the context variable) may change, but the dynamics of the world do not. For example, the context variable could be a continuous variable like a goal position that the expert is navigating towards or a discrete variable representing a behavior like locking a door.

Data generation. Data is generated by executing policies π that map observations to actions in the environment. We consider two different datasets for any given problem. First there is a large multi-context pretraining dataset that will be used for representation learning, specifically to learn an observation encoder. Second, there is a small single-context finetuning dataset for policy learning on top of the pretrained representation. The multi-context pretraining data is generated as follows:

$$D_{pre} = \{\tau_i\}_{i=1}^{N_{pre}} : \quad c \sim P_c, \quad \tau = (o_0, a_0, o_1 \dots) \sim P^{\pi_c}, \quad \pi_c \approx \pi_c^* = \arg\max_{\pi} J_{r_c}(\pi), \tag{6.2}$$

where $J_{r_c}(\pi)$ denotes the expected return of π when the reward is r_c. Note that the demonstration policy has access to the latent context c, but this latent context is not observed in the data.

Then the single-context finetuning data is generated for context c_{fine} as follows:

$$D_{fine} = \{\tau_i\}_{i=1}^{N_{fine}} : \tau = (o_0, a_0, o_1 \dots) \sim P^{\pi_{c_{fine}}}. \tag{6.3}$$

Pretraining. The goal of the paper is to analyze different methods for pretraining feature extractors. Training of the encoders ϕ to minimize a loss ℓ proceeds as follows:

$$\hat{\phi} : \mathcal{O} \to \mathbb{R}^d = \arg\min_{\phi} \mathop{\mathbb{E}}_{D_{pre}} [\ell(\phi, \tau_i)]. \tag{6.4}$$

A full description of the losses ℓ used by different algorithms will come in Section 6.4.2. For simplicity (and in keeping with prior work [Nachum and Yang 2021; Chen et al. 2022]) we will consider ℓ to only be a function of transitions $(o_i^j, a_i^j, o_i^{j'})$ rather than full trajectories to leverage the Markovian structure. We also run some ablations of including multistep information in Appendix D.2 and find little difference.

Finetuning. Features are evaluated by finetuning a small policy head on top of the frozen features:

$$\hat{\pi}_{\hat{\phi}} : \mathbb{R}^d \to \mathcal{A} = \arg\min_{\pi} \mathop{\mathbb{E}}_{D_{fine}} [\ell(\pi, a_i^j, \hat{\phi}(o_i^j))]. \tag{6.5}$$

We elect to use frozen features to allow for simple and clear evaluation of the representations. This is in keeping with prior work on representations for imitation [Nachum and Yang 2021; Chen et al. 2020a; Nair et al. 2022] as well as computer vision [Chen et al. 2020a].

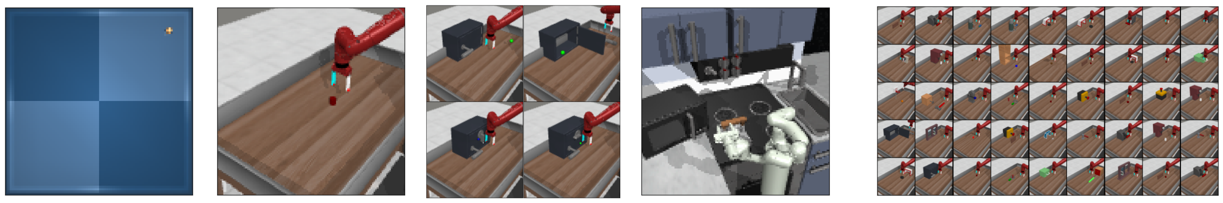
Evaluation. Finally, we evaluate the finetuned policy by performing rollouts in the finetuning environment with context c_{fine} to estimate $J_{r_{c_{fine}}}(\hat{\pi}_{\hat{\phi}})$. In our tasks we usually consider $r_{c_{fine}}$ to be a binary indicator of successful completion of the finetuning task.

6.4 Experimental setup

6.4.1 Environments and Datasets

We design a suite of tasks and datasets to probe the capabilities of various representation learners for downstream imitation. We focus on robotic manipulation from vision as this is an important sequential decision making task that depends on learning task-relevant visual representations where pretraining deep visual feature extractors is a popular approach. Our suite consists of six different pretraining datasets on varied tasks and of varied size. Each pretraining dataset has several associated finetuning datasets and simulation environments that allow for online evaluation of learned policies.

All tasks are performed from visual inputs, as shown in Figure 6.2. Each pair of pretraining-finetuning datasets requires a slightly different type of generalization as dictated by the different



(a) Pointmass **(b)** Pick + place **(c)** Door **(d)** Kitchen **(e)** Metaworld (ML45/R3M)

Figure 6.2: Our six datasets: (a) Pointmass navigation with latent goals. (b) Pick and place with latent goals. (c) Multitask manipulation of a door. (d) Sequential kitchen manipulation. (e) Multitask manipulation of diverse objects, where we consider two different train-eval splits ML45 and R3M.

types of context variable and is described in detail in Appendix D.3.

Table 6.1: Description of the different datasets used in the experiments. Dataset sizes are measured in number of trajectories (N^{pre}_{traj} for pretraining and N^{fine}_{traj} for finetuning) and given as ranges with default values in bold. Trajectory lengths vary from 50 to 400 steps. These default sizes may vary in each experiment when indicated. Each datasets contains a certain number of latent contexts ($N^{pre}_{context}$ and $N^{fine}_{context}$). For each finetuning context, we sample datasets with N^{fine}_{seed} different seeds.

Environment	N^{pre}_{traj}	N^{fine}_{traj}	$N^{pre}_{context}$	$N^{fine}_{context}$	N^{fine}_{seed}
Pointmass	(1e1, 1e2, **1e3**)	(1, **2**, 5, 10)	N^{pre}_{traj}	5	1
Pick + place	(1e1, 1e2, **1e3**)	(2, **5**, 10, 20)	N^{pre}_{traj}	5	1
Door	(1e1, 1e2, **1e3**)	(2, 5, **10**, 20)	3	1	5
Kitchen	(50, 150, **450**)	(2, 5, **10**, 15)	21	3	5
MW-ML45	(1e2, 1e3, **1e4**)	(2, 5, **10**, 20)	45	5	5
MW-R3M	(1e2, 1e3, **1e4**)	(2, 5, **10**, 20)	45	5	5

6.4.2 Algorithms

We consider nine different representations across our suite of experiments. These representations include baseline and skyline/oracle performance as well as five representations that are pretrained on our own pretraining datasets described above. Each of the representations will be referred to by its bolded name after it is described.

All algorithms (except for the Imagenet and R3M baselines) share the exact same encoder architecture to control as best we can for variation in architecture between methods. Each method is pretrained for the same number of gradient steps. Additional training details can be found in Appendix D.3.

Skyline/oracle. As a skyline or oracle representation we directly use the low dimensional states (**States**) from the simulator. Depending on the task, this representation includes the position of the robot, position of the object to be manipulated, and/or position of the goal. A full description of the per environment state variables can be found in Appendix D.3.

Baselines. We consider three baseline representations that are not trained on our pretraining datasets. The first is to directly use the pixels with image augmentations (**Pixels + Aug**) to train an encoder and a policy from scratch on the finetuning data. It is essential to use the augmentations to ensure that this a strong baseline. The second is features of a ResNet18 pretrained on Imagenet (**Imagenet**). The last consists of the features of a ResNet18 that is specifically pretrained for robotic manipulation by [Nair et al. 2022] on the Ego4d dataset (**R3M**).

Inverse dynamics. The primary representation learning objective that we consider is inverse dynamics (**ID**) which models the distribution $P(a|o, o')$ using an architecture that first encodes o, o' with an encoder ϕ and then predicts a with a small MLP f:

$$\phi^*_{ID} = \arg\min_{\phi} \min_{f} \mathop{\mathbb{E}}_{o,a,o'} [(a - f(\phi(o), \phi(o')))^2]. \tag{6.6}$$

Behavior cloning. A simpler alternative objective is to directly apply behavior cloning (**BC**) to the multitask actions in the pretraining dataset conditioned on the observations using MSE loss. The learner is parameterized as an encoder ϕ followed by a small MLP π:

$$\phi^*_{BC} = \arg\min_{\phi} \min_{\pi} \mathop{\mathbb{E}}_{o,a} [(a - \pi(\phi(o)))^2]. \tag{6.7}$$

Forward dynamics. We consider two representation learners that predict the forward dynamics of the system. The first is explicit forward dynamics (**FD-e**) which explicitly constructs a model of the forward dynamics in the space of observations by encoding the current observation and then attempting to reconstruct the next observation o' using a decoder d:

$$\phi_{EFD}^* = \arg\min_{\phi} \min_{d} \mathbb{E}_{o,a,o'} [(o' - d(\phi(o), a))^2]. \tag{6.8}$$

The second objective is implicit forward dynamics (**FD-i**) which implicity constructs a model of the forward dynamics using contrastive learning. Explicitly, we consider a form of contrastive learning where an energy function is defined as the inner product of L2-normalized projected embeddings (given by projection MLPs f_1, f_2) which is then passed into an InfoNCE loss:

$$E(o, a, o') = \exp(f_1(\phi(o), a)^\top f_2(\phi(o'))), \tag{6.9}$$

$$\phi_{IFD}^* = \arg\min_{\phi} \min_{f_1, f_2} \mathbb{E}_{o,a,o'} [-\log(E(o, a, o')) + \log \mathbb{E}_{\bar{o}'} [E(o, a, \bar{o}')]]. \tag{6.10}$$

Static observation modeling Finally, we consider a baseline that simply models $P(o)$. Rather than modeling this explicitly with reconstruction, we use a contrastive loss (**Cont**) where we use image augmentations to construct pairs of o and $\bar{o}$ that do not rely on the dynamics of the environment at all. Again we use the InfoNCE loss, in what can be seen as a variant of SimCLR:

$$E(o, o_{aug}) = \exp(f(\phi(o))^\top \pi(\phi(o_{aug}))), \tag{6.11}$$

$$\phi_{Cont}^* = \arg\min_{\phi} \min_{f} \mathbb{E}_{o, o_{aug}} [-\log(E(o, o_{aug})) + \log \mathbb{E}_{\bar{o}_{aug}} [E(o, \bar{o}_{aug})]]. \tag{6.12}$$

6.5 Experiments

We want to determine which representation learning objective is best, but the precise answer will depend on the situation. To get a clearer understanding of this sometimes ambiguous performance we conduct a variety of controlled experiments on our diverse suite of datasets. We focus on the following questions to guide our empirical analysis:

1. How do factors of the datasets impact performance of algorithms?
2. How are the learned representations similar to and different from each other?

Note: we will focus on presenting aggregate statistics across all datasets in the main text, but full results can be found in Appendix D.2. Full details about the methodology are in Appendix D.3 and code is at `https://github.com/davidbrandfonbrener/imitation_pretraining`.

6.5.1 Impact of dataset on representation learning performance

Scaling with data size. The performance of each algorithm can be highly sensitive to both pretraining and finetuning sizes. Thus, instead of producing one simple summary statistic, we sweep over both the size of the finetuning data (for default pretraining size) and size of the pretraining data (for default finetuning size). The results of these sweeps are presented in Figure 6.3.

The sweeps both suggest that inverse dynamics outperforms the alternatives. First, on the finetuning size sweep, we see that the ID line is the only one that consistently outperforms training from scratch on Pixels + Aug. This gap is largest at small finetuning sizes, which are perhaps the most interesting case since that is when we expect pretraining to be useful. Second, the pretraining size sweep indicated that ID is scaling the most efficiently with pretraining size. Further results, including breakdowns across each dataset can be found in Appendix D.2.

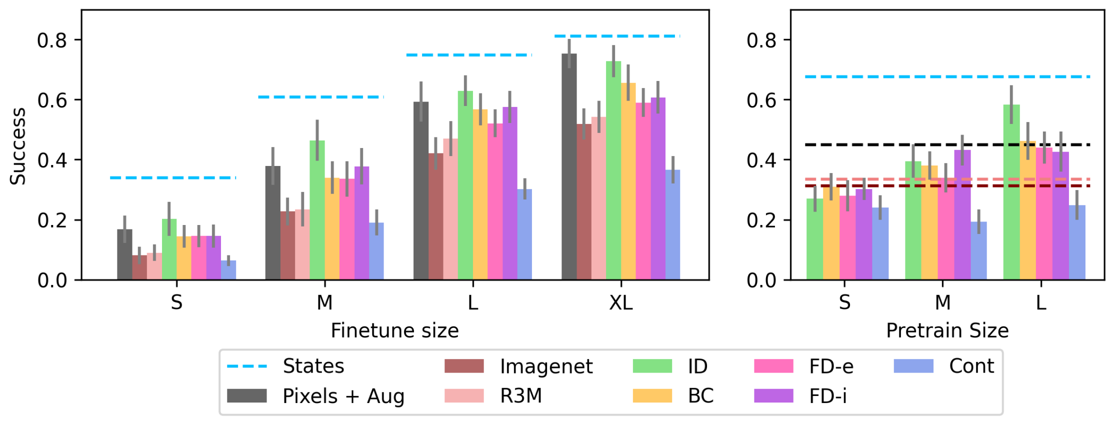


Figure 6.3: Average success rate after finetuning averaged across datasets, contexts, and seeds. Error bars show the standard error across contexts and seeds, averaged across datasets. The plots show sweeps across finetuning size with default pretraining size (left) or pretraining size with default finetuning size (right) measured in units according to Table 6.1. Methods that do not depend on pretraining size are shown as horizontal lines.

In distribution vs. out of distribution eval tasks. The way that our datasets are constructed, the door, kitchen, metaworld-ml45, and metaworld-r3m datasets only have a finite number of possible contexts that is much smaller than the number of pretraining trajectories. For our default datasets, we elected to construct a train-test split of contexts to ensure that the contexts used for finetuning are not seen during pretraining. As a result, the default finetuning tasks can be in some sense "out of distribution", measuring extrapolation as opposed to in-distribution generalization. For example, in the door dataset, we pretrain on door opening, closing, and unlocking (with varied door position) and then finetune on door locking (again with varied position).

To test the impact of this gap between pretraining and finetuning, we created alternative pretraining datasets, where we include the test contexts (but *not* the test trajectories) into the pretraining data. For example, in the door domain we include door opening, closing, locking, *and* unlocking in the pretraining data and still finetune on only unlocking (but with heldout initial conditions). These datasets now require a much more limited notion of generalization from pretraining to finetuning.

Results are shown in Figure 6.4. We again see that ID is the strongest performer, but now the gap is even larger. ID matches the skyline performance of training from ground truth low-dimensional

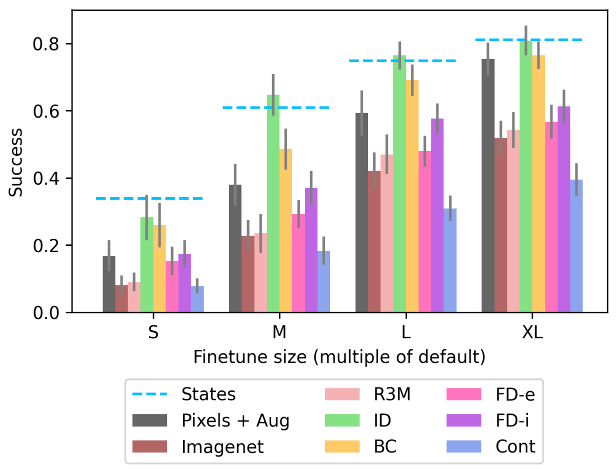


Figure 6.4: Average performance on the four discrete context environments when the finetuning contexts are included in the pretraining data. The finetuning data contains heldout initial conditions and trajectories not seen during pretraining.

simulator states. BC also shows substantially stronger performance and outperfroms training from scratch Pixels + Aug. None of the other pretraining algorithms benefit much from the substantially easier type of generalization required on these datasets. This suggests that ID and BC are uniquely able to benefit in easier settings, suggesting that they are better representation learners. If an algorithm is not able to outperform training from scratch in this simplified setting, it is unlikely to be a good representation learner.

Fully latent vs. inferrable context variables. Looking at our dataset suite, the datasets can be divided into two groups: those where the context variable is not inferrable at all from the initial state (pointmass, pick+place, and kitchen), and those where the effect of the context variable on the initial state makes it possible to infer the context given the initial state (door, metaworld-ml45, and metaworld-r3m). This split presents an interesting comparison in particular between ID and BC (the best performing algorithms from the prior experiment). Figure 6.5 shows results for these algorithms and the Pixels + Aug baseline on the datasets where the context is latent.

There is a large gap between ID and BC when the context is fully latent. In these cases, it is

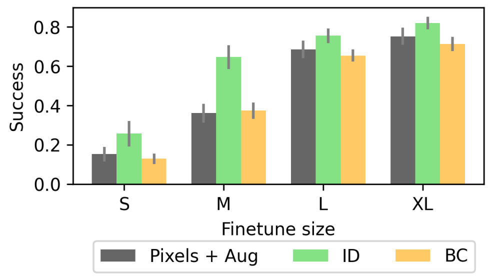


Figure 6.5: A comparison between ID and BC on the datasets where the context is not inferrable from the observation.

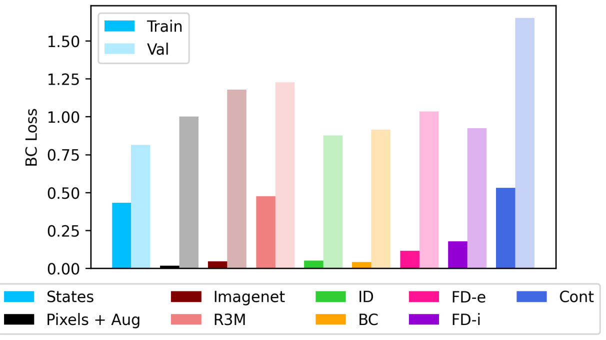


Figure 6.6: Average train and validation action-prediction loss during finetuning. All losses are normalized by the Pixels + Aug validation loss to maintain consistency across environments.

impossible to tell from the current state alone what the context is and thus what the optimal action should be. As we will show in our simplified model (Section 6.6), in these settings BC is *confounded* by the latent context (in the terminology of causal inference). As a result, BC can fail to learn useful features. In contrast, ID uses the information about the future state to deconfound the learning problem and still learns a good representation. Note that this gap largely disappears when the context is observable, see Appendix D.2 for further details.

6.5.2 Predictive power of the representations

So far, we have focused on the success rate of the downstream finetuned policy as the main metric of comparison between algorithms. Now we will instead consider a series of experiments that assess the quality of the representations based on the ability to predict various quantities of interest from the representations. These experiments help to illustrate what information is retained in the representations and how efficiently that information can be accessed.

Action prediction. First, we consider the ability to predict the expert actions in the finetuning dataset. This is directly related to the success of the finetuned policy, but avoids the variance of performing rollouts and allows us to compare train and validation errors to evaluate the representations. Low train loss means the representations are not aliasing observations that require different actions. Meanwhile the validation loss measures the simplicity of the function that maps representations to targets, i.e. how well it generalizes.

The results in Figure 6.6 show the train and validation loss during finetuning using the default pretraining and finetuning sizes from Table 6.1. Since losses vary across datasets, we normalize by the Pixels + Aug validation loss so as to be able to present averages across all datasets. We see that out of the learned representations, ID has both the lowest train and validation losses, almost matching the performance of Pixels + Aug on train and almost matching the performance of States on validation. In contrast, representations that attempt to predict forward dynamics have substantially higher train loss, indicating aliasing of states in terms of their optimal actions. Interestingly, the Imagenet pretrained features have very low train loss, indicating a lack of aliasing, but very high validation loss, indicating that the function that maps representations to actions does not generalize well.

State prediction. Since we perform all of our experiments in simulated environments, we have access the the ground truth low dimensional states. So, we can measure the ability of each representation to predict the ground truth low dimensional state and thus measure how well the representation retains information about this ground truth state. Results are in Figure 6.7; here we measure the train and validation loss on the pretraining distribution so as to isolate the effect of the representation learning apart from the gap between pretraining and finetuning. Again we

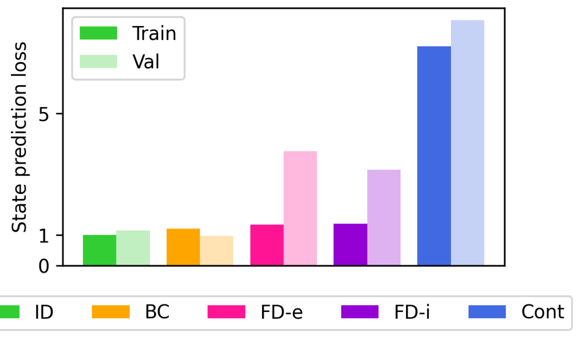


Figure 6.7: Average state prediction error on the pretraining distribution. Values are normalized by the ID train loss.

normalize the losses for each dataset.

Again we see that ID and BC yield the best performance. This suggests that in these datasets, pretraining objectives that attempt to predict the optimal action do indeed facilitate recovery of the low-dimensional simulator state. In contrast, while the FD methods achieve approximately the same training error, they generalize much more poorly. This suggests that the FD objectives are not throwing away relevant information, but are keeping around too much extraneous information about the observations, thus making the representations susceptible to overfitting. Standard contrastive learning is substantially worse, even on train error, suggesting that it is throwing away important information. Extended results are in Appendix D.2.

6.6 Analysis

To add a more theoretical understanding of the empirical results, we will consider a simplified model of the data generating process based on linear dynamics in a latent space. We begin by presenting the model and then show that under this model we can explain three key experimental findings: (1) inverse dynamics is able to recover the low dimensional state, (2) forward dynamics can be less efficient in some cases, and (3) BC can be confounded by the latent context. We present a high level sketch here and more details along with discussion of related theoretical work are in

Appendix D.4.

Model. Some of the key interesting properties of problems like visual manipulation that we consider empirically are that (a) the observation is very high dimensional relative to the action, (b) the actual state of the world (or simulator) can be summarized in a much lower dimensional state variable, and (c) the dynamics are relatively simple if given the right representation. All of these motivating properties can be captured in a simplified model that assumes linear dynamics occurring in a hidden low-dimensional state space, as presented below.

For simplicity, we will only consider one step of the dynamics represented by a tuple (o, a, o', s, s', c) that is sampled iid from the joint distribution over those variables. Recall that we only observe (o, a, o') and that (s, s', c) are latent. Formally, let $\mathcal{O} = \mathbb{R}^d$, $\mathcal{S} = \mathbb{R}^\ell$, and $\mathcal{A} = \mathbb{R}^k$ with $d \gg \ell > k$. Let $\phi : \mathcal{O} \to \mathcal{S}$ be the ground truth encoder, which we assume is invertible by ϕ^{-1}. Let $\epsilon \sim \mathcal{N}(0, \Sigma)$ in $\mathbb{R}^\ell$ and A, B to be any matrices in $\mathbb{R}^{\ell\times\ell}$ and $\mathbb{R}^{\ell\times k}$. Then, assume that the dynamics are:

$$o' = \phi^{-1}(A\phi(o) + Ba + \epsilon). \tag{6.13}$$

Note that we make no assumption on the policy π_c^* other than that it only depends on o via $\phi(o)$. This model is similar to ones studied in the online control setting by Mhammedi et al. [2020]; Dean and Recht [2021], but is different from models where inverse dynamics have been studied for online control with exogenous noise since the dynamics are entirely contained in the low dimensional state space [Efroni et al. 2021; Lamb et al. 2022].

Inverse dynamics recovers the state. To get an intuition as to why inverse dynamics learning is feasible in this model, note that if B^+ is the pseudoinverse of B that:

$$a = B^+\phi(o') - B^+A\phi(o) - B^+\epsilon. \tag{6.14}$$

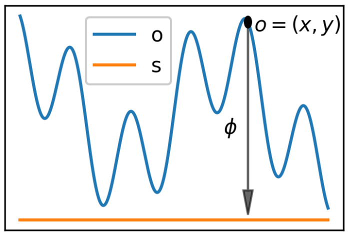


Figure 6.8: An example where the decoder is more complicated than the encoder.

Thus the inverse dynamics are a simple linear function of the embeddings $\phi(o), \phi(o')$. As a result, when we solve for a with least squares regression, if the encoder ϕ is representable by our function class, we will be able to recover it up to linear transformation, provided the matrix B is well-conditioned, so that the noise term $B^{+}\epsilon$ does not blow up.

Forward dynamics can be less statistically efficient. Intuitively, the potential problem with learning forward dynamics is that it requires learning both an encoder *and* a decoder while inverse dynamics *only* requires learning the encoder. This is not necessarily a problem a priori, but we hypothesize that in practical problems of interest (like the ones in our experiments) the decoder (mapping from low dimensional state to high dimensional observation) may be more complicated than the encoder (mapping from observations to states).

To grasp why we might expect this, note that the set of possible observations is the manifold represented by the image of the decoder, i.e. $\mathrm{Im}(\phi^{-1})$. As a simple example, consider a toy 2d example where the high dimensional observation is $(x, f(x)) \in \mathbb{R}^2$ and the low dimensional state is simply $x \in \mathbb{R}^1$, as depicted in Figure 6.8. Here the encoder ϕ is very simple since it just needs to recover x, while the decoder must learn $f(x)$. Of course this is a very toy example, but we find it illustrative of the idea that it is possible that the encoder is much simpler than the decoder in practice.

BC can be confounded by the latent context. As we alluded to in the experimental section, the latent context variable can confound BC. Now we will show an example in our model where this problem arises. In this case, even with a linear encoder, infinite data, and a fully expressive policy class, the Bayes optimal BC representation cannot be used to recover anything better than a random policy. This example is extreme, but shows the shortcomings of a confounded pretraining objective.

For simplicity, let $\ell = k$ and $\epsilon = 0$. Let $\mathcal{R}(\mathbb{R}^{k\times k})$ be the set of rotation matrices in $\mathbb{R}^k$. Let $\mathbb{S}^{k-1}$ be the unit sphere in $\mathbb{R}^k$, U be the uniform distribution, and δ denote a Dirac delta. Now, assume:

$$c \sim U(\mathcal{R}(\mathbb{R}^{k\times k})), \quad o \sim U(\phi^{-1}(\mathbb{S}^{k-1})), \quad \pi_c^*(a|o) = \delta[a = c\phi(o)] \tag{6.15}$$

Note that $\phi(o)$ returns a unit vector in $\mathbb{R}^k$ and that a uniformly sampled rotation of a unit vector is a uniformly sampled unit vector. Thus, we can marginalize over c to get:

$$P(a|o) = \int_c P(c)\pi_c^*(a|o) = \int_c P(c)\delta[a = c\phi(o)] = P_{U(\mathbb{S}^{k-1})}(a) = \eta_k, \tag{6.16}$$

for a constant η_k equal to the reciprocal of the surface area of the unit sphere in $\mathbb{R}^k$.

Thus, the Bayes optimal BC policy does not depend on o at all. As a result, the optimal representation learned by BC can just map every observation to zero. This representation is not capable of representing the optimal policy for any choice of c. However, switching to inverse dynamics pretraining where we condition on the outcome observation o' breaks the confounding and allows us to learn the true representation even without observing c.

6.7 Discussion

We have seen that inverse dynamics pretraining provides an effective method for learning features from multitask demonstration data. We demonstrated this across a suite of datasets with visual observations and provided analysis in a simplified model to understand the strong empirical performance. Going forward, there are many interesting directions for future work such as: scaling up inverse dynamics pretraining to larger real world tasks, going beyond the imitation setting to consider learning from suboptimal data or in online settings, and comparing against pretraining techniques that go beyond feature learning (such as goal conditioned and reward-based policies).

7 | Discussion

This thesis has presented a variety of work that attempts to bridge the gap between supervised learning and control. In Part I we considered simplified algorithmic approaches to the offline reinforcement learning problem that attempt to make the algorithms more similar to standard supervised learning. While these algorithms have clear limitations in terms of attempting to recover the optimal policy, they also have clear benefits in terms of stabilized and simplified training that can make them useful components of broader systems for solving control problems.

In Part II we instead focused on the data for offline reinforcement learning, which is the limiting factor in how well we can hope for any algorithm to do. We consider two extreme goals of data collection, either maximal exploration without any supervision (Chapter 4) or maximal supervision, i.e. expert teleoperation (Chapter 5). Both data collection scenarios raise interesting questions and opportunities. The exploratory data has the possibility of solving many tasks from one dataset, but is difficult to scale to hard-exploration tasks. Collecting datasets via teleoperation can lead to good data efficiency and task success, but is limited by the capabilities and cost of the teleoperation. Both chapters illustrate how thinking deeply about the data that is fed into a learning algorithm can potentially yield large returns.

Finally, in Part III we consider how to use self-supervised pretraining approaches for downstream control. Chapter 6 presents results in a multitask imitation setting where we find that modeling the inverse dynamics as a self-supervised pretraining task works better than the alternatives. This work takes a first step towards bringing scalable self-supervised pretraining into control problems where the datasets are often more structured (containing trajectories rather than single datapoints) than in standard supervised settings, which opens the door for more targeted pretraining objectives.

There are of course many interesting directions for future work to continue to bridge the gap from supervised learning to control. In particular, the understanding of how dataset collection can impact downstream reinforcement learning is still understudied. We presented preliminary work studying fully exploratory (Chapter 4) and fully expert (Chapter 5) data collection, but there is clearly a lot of room in between these extremes that is likely to be very useful in practice. For example, how can we use experts to guide effective exploration? Another important theoretical direction is to get a better understanding of what is possible and what is optimal in terms of data collection. With a fixed budget for collecting data, what sort of data should we be attempting to collect? The VBT approach presents a heuristic answer to this question, but a more rigorous answer could yield interesting practical insights.

Another interesting possible direction to build on the results in this thesis would be to extend the ideas about pretraining presented in Part III. That chapter considers standard transition-level pretraining objectives that have been studies in the literature before. Those transition-level objective leverage the Markov structure of the problem, but they forgo much of the recent advances in scalable sequence modeling with transformers. Some recent work has instead pretrained by modeling the full sequence [Wu et al. 2023; Chen et al. 2021b; Yang and Nachum 2021]. This sort of pretraining may look more similar to language modeling, which has recently seen much success for natural language processing [Brown et al. 2020]. Perhaps the most interesting property of the recent advances in large language models is that the models can be effectively prompted to accomplish a massively diverse set of tasks without needing to finetune the weights. An interesting future direction is to pretrain sequence models for control and to leverage prompting as well as finetuning to effectively adapt to new problems. As we saw in Chapter 3, to effectively solve control problems, the prompting scheme likely needs to go beyond simply conditioning on returns. Instead conditioning on language or sequences of states could yield richer generalization. Of course, this approach will still run up against all of the issues with collecting broad and interesting datasets for control problems discussed elsewhere in this thesis.

More speculatively, there is also an opportunity to bridge the gap in the other direction, by bringing tools and ideas from control and reinforcement learning into regimes that are traditionally approached with supervised learning alone. Large language models (LLMs) are a particularly interesting case here. For starters, LLMs are already being finetuned with reinforcement learning from human feedback because, while the models may be pretrained with supervised learning, the ultimate use case is to have easily controllable generation which starts to look more like a control problem (albeit with a somewhat ill-specified reward). But perhaps more interesting are directions thinking about how to more actively collect and filter the data that models are trained on, an issue that is often central for control problems. For example, the distribution of data observed in deployed systems will be shaped by (and maybe shape) the distribution of text supplied by users. Or alternatively, even the selection of pretraining data itself is now often shaped by intuitive heuristics, but motivated by ideas like those discussed in Part II, it would be interesting to think about how to shape the collection and curation of these datasets in automated ways to facilitate downstream goals.

In sum, this thesis has presented several pieces of works that attempt to bridge the gap between supervised learning and control, but there is clearly still much work to be done to leverage the tools from these two subfields going forward.

APPENDIX A

Appendix for Offline RL without off-policy evaluation

A.1 Gridworld example where multi-step outperforms one-step

As explained in the main text, this section presents an example that is only a slight modification of the one in Figure 2.4, but where a multi-step approach is clearly preferred over just one step. The data-generating and learning processes are exactly the same (100 trajectories of length 100, discount 0.9, $\alpha = 0.1$ for reverse KL regularization). The only difference is that rather than using a behavior that is a mixture of optimal and uniform, we use a behavior that is a mixture of maximally suboptimal and uniform. If we call the suboptimal policy π^- (which always goes down and left in our gridworld), then the behavior for the modified example is $\beta = 0.2 \cdot \pi^- + 0.8 \cdot u$, where u is uniform. Results are shown in Figure A.1.

By being more likely to go to the noisy states, this behavior policy allows us to get lower variance estimates of the rewards. Essentially, the coverage of the behavior policy in this example reduces

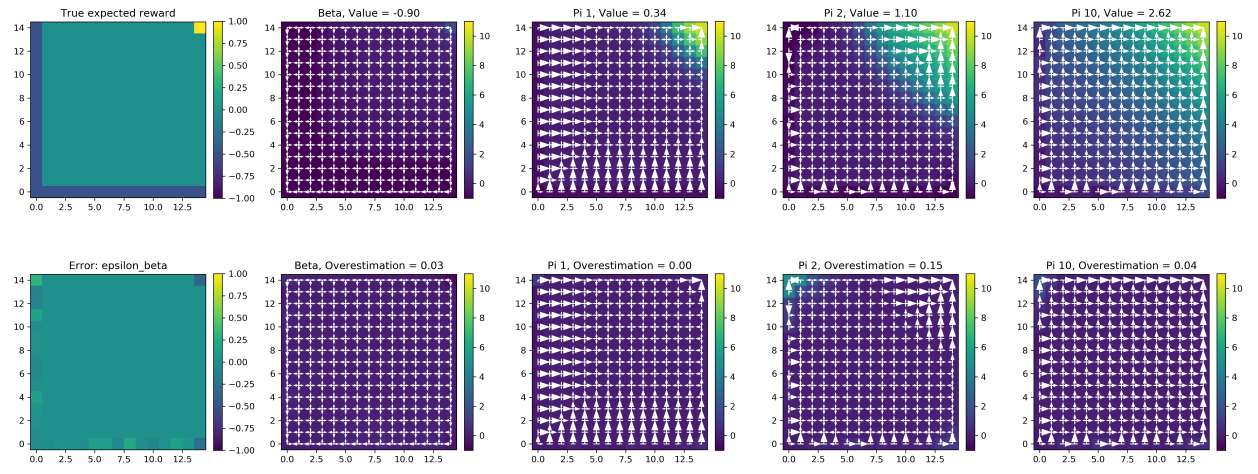


Figure A.1: A gridworld example with modified behavior where multi-step is much better than one-step.

the magnitude of the evaluation errors. This allows for more aggressive planning using multi-step methods. Moreover, since the behavior is less likely to go to the good state, the behavior Q function does not propagate the signal from the rewarding state as far, harming the one-step method.

A.2 Connection to policy improvement guarantees

The regularized or constrained one-step algorithm performs an update that directly inherits guarantees from the literature on conservative policy improvement [Kakade and Langford 2002; Schulman et al. 2015; Achiam et al. 2017]. These original papers consider an online setting where more data is collected at each step, but the guarantee at each step applies to our one-step offline algorithm.

The key idea of this line of work begins with the performance difference lemma of [Kakade and Langford 2002], and then lower bounds the amount of improvement over the behavior policy. Define the discounted state visitation distribution for a policy π by $d^\pi(s) := (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \mathbb{P}_{\rho,P,\pi}(s_t = s)$. We will also use the shorthand $Q(s,\pi)$ to denote $\mathbb{E}_{a\sim\pi|s}[Q(s,a)]$. Then we have the performance difference lemma as follows.

Lemma A.1 (Performance difference, [Kakade and Langford 2002]). *For any two policies π and β,*

$$J(\pi) - J(\beta) = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s \sim d^\pi} [Q^\beta(s, \pi) - Q^\beta(s, \beta)]. \tag{A.1}$$

Then, Corollary 1 from [Achiam et al. 2017] (reproduced below) gives a guarantee for the one-step algorithm. The key idea is that when π is sufficiently close to β, we can use Q^β as an approximation to Q^π.

Lemma A.2 (Conservative Policy Improvement, [Achiam et al. 2017]). *For any two policies π and β, let $\|A_\pi^\beta\|_\infty = \sup_s |Q^\beta(s, \pi) - Q^\beta(s, \beta)|$. Then,*

$$J(\pi) - J(\beta) \geq \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s \sim d^\beta} \left[\left(Q^\beta(s, \pi) - Q^\beta(s, \beta) \right) - \frac{2\gamma \|A_\pi^\beta\|_\infty}{1-\gamma} D_{TV}(\pi(\cdot|s) \| \beta(\cdot|s)) \right] \tag{A.2}$$

where D_{TV} denotes the total variation distance.

Replacing Q^β with $\widehat{Q}^\beta$ and the TV distance by the KL (using Pinsker's inequality), we get precisely the objective that we optimize in the one-step algorithm. This shows that the one-step algorithm indeed optimizes a lower bound on the performance difference. Of course, in practice we replace the potentially large multiplier on the divergence term by a hyperparameter, but this theory at least motivates the soundness of the approach.

We are not familiar with similar guarantees for the iterative or multi-step approaches that rely on off-policy evaluation.

A.3 Experimental setup

Code for our experimental setup can be found at `https://github.com/davidbrandfonbrener/onestep-rl`.

Table A.1: Hyperparameter sweeps for each algorithm.

Algorithm	Hyperparameter set
Reverse KL (α)	{0.03, 0.1, 0.3, 1.0, 3.0, 10.0}
Easy BCQ (M)	{2, 5, 10, 20, 50, 100}
Exponentially weighted (τ)	{0.1, 0.3, 1.0, 3.0, 10.0, 30.0}

A.3.1 Benchmark experiments (Tables 2.1 and 2.2, Figure 2.2)

Data. We use the datasets from the D4RL benchmark [Fu et al. 2020]. We use the latest versions, which are v2 for the mujoco datasets and v1 for the adroit datasets.

Hyperparameter tuning. We follow the practice of [Fu et al. 2020] and tune a small set of hyperparameters by interacting with the simulator to estimate the value of the policies learned under each hyperparameter setting. The hyperparameter sets for each algorithm can be seen in Table A.1. We tune hyperparameters using 3 seeds, but then evaluate the best hyperparameter by training on an additional 7 seeds and then report results on the 10 total seeds.

This may initially seem like “cheating”, but can be a reasonable setup if we are considering applications like robotics where we can feasibly test a small number of trained policies on the real system. Also, since prior work has used this setup, it makes it easiest to compare our results if we use it too. While beyond the scope of this work, we do think that better offline model selection procedures will be crucial to make offline RL more broadly applicable. A good primer on this topic can be found in [Paine et al. 2020].

Models. All of our Q functions and policies are simple MLPs with ReLU activations and 2 hidden layers of width 1024. Our policies output a truncated normal distribution with diagonal covariance where we can get reparameterized samples by sampling from a uniform distribution and computing the differentiable inverse CDF [Burkhardt 2014]. We found this to be more stable than the tanh of normal used by e.g. [Fu et al. 2020], but to achieve similar performance when both are stable. We use these same models across all experiments.

One-step training procedure. For all of our one-step algorithms, we train our $\hat{\beta}$ behavior estimate by imitation learning for 500k gradient steps using Adam [Kingma and Ba 2014] with learning rate 1e-4 and batch size 512. We train our $\widehat{Q}^{\beta}$ estimator by fitted Q evaluation with a target network for 2 million gradient steps using Adam with learning rate 1e-4 and batch size 512. The target is updated softly at every step with parameter $\tau = 0.005$. All policies are trained for 100k steps again with Adam using learning rate 1e-4 and batch size 512.

Easy BCQ does not require training a policy network and just uses $\hat{\beta}$ and $\widehat{Q}^{\beta}$ to define it's policy. For the exponentially weighted algorithm, we clip the weights at 100 to prevent numerical instability. To estimate reverse KL at some state we use 10 samples from the current policy and the density defined by our estimated $\hat{\beta}$.

Each random seed retrains all three models (behavior, Q, policy) from different initializations. We use three random seeds.

Multi-step training procedure. For multi-step algorithms we use all the same hyperparameters as one-step. We initialize our policy and Q function from the same pre-trained $\hat{\beta}$ and $\widehat{Q}^{\beta}$ as we use for the one-step algorithm trained for 500k and 2 million steps respectively. Then we consider 5 policy steps. To ensure that we use the same number of gradient updates on the policy, each step consists of 20k gradient steps on the policy followed by 200k gradient steps on the Q function. Thus, we take the same 100k gradient steps on the policy network. Now the Q updates are off-policy so the next action a' is sampled from the current policy π_i rather than from the dataset.

Iterative training procedure. For iterative algorithms we again use all the same hyperparameters and initialize from the same $\hat{\beta}$ and $\widehat{Q}^{\beta}$. We again take the same 100k gradient steps on the policy network. For each step on the policy network we take 2 off-policy gradient steps on the Q network.

Evaluation procedure. To evaluate each policy we run 100 trajectories in the environment and compute the mean. We then report the mean and standard error over 10 training seeds.

A.3.2 MSE experiment (Figure 3)

Data. To get an independently sampled dataset of the same size as the training set, we use the behavior cloned policy $\hat{\beta}$ to sample 1000 trajectories. The checkpointed policies are taken at intervals of 5000 gradient steps from each of the three training seeds.

Training procedure. The $\widehat{Q}^{\pi_i}$ training procedure is the same as before so we use Adam with step size 1e-4 and batch size 512 and a target network with soft updates with parameter 0.005. We train for 1 million steps.

Evaluation procedure. To evaluate MSE, we sample 1000 state, action pairs from the original training set and from each state, action pair we run 3 rollouts. We take the mean over the rollouts and then compute squared error at each state, action pair and finally get MSE by taking the mean over state, action pairs. The reported reverse KL is evaluated by samples during training. At each state in a batch we take 10 samples to estimate the KL at that state and then take the mean over the batch.

A.3.3 Gridworld experiment (Figure 4)

Environment. The environment is a 15 x 15 gridworld with deterministic transitions. The rewards are deterministically 1 for all actions taken from the state in the top right corner and stochastic with distribution $\mathcal{N}(-0.5, 1)$ for all actions taken from states on the left or bottom walls. The initial state is uniformly random. The discount is 0.9.

Data. We collect data from a behavior policy that is a mixture of the uniform policy (with probability 0.8) and an optimal policy (with probability 0.2). We collect 100 trajectories of length 100.

Training procedure. We give the agent access to the deterministic transitions. The only thing for the agent to do is estimate the rewards from the data and then learn in the empirical MDP. We perform tabular Q evaluation by dynamic programming. We initialize with the empirical rewards and do 100 steps of dynamic programming with discount 0.9. Regularized policy updates are solved for exactly by setting $\pi_i(a|s) \propto \beta(a|s)\exp(\frac{1}{\alpha}\widehat{Q}^{\pi_{i-1}}(s,a))$.

A.3.4 Overestimation experiment (Figure 5)

This experiment uses the same setup as the MSE experiment. The main difference is we also consider the Q functions learned during training and demonstrate the overestimation relative to the Q functions trained on the evaluation dataset as in the MSE experiment.

A.3.5 Mixed data experiment (Figure 6)

We construct datasets with $p_m = \{0.0, 0.1, 0.2, 0.4, 0.6, 0.8, 1.0\}$ by mixing the random and medium datasets from D4RL and then run the same training procedure as we did for the benchmark experiments. Each dataset has the same size, but a different proportion of trajectories from the medium policy.

A.4 Learning curves

In this section we reproduce the learning curves and hyperparameter plots across the one-step, multi-step, and iterative algorithms with reverse KL regularization, as in Figure 2.2.

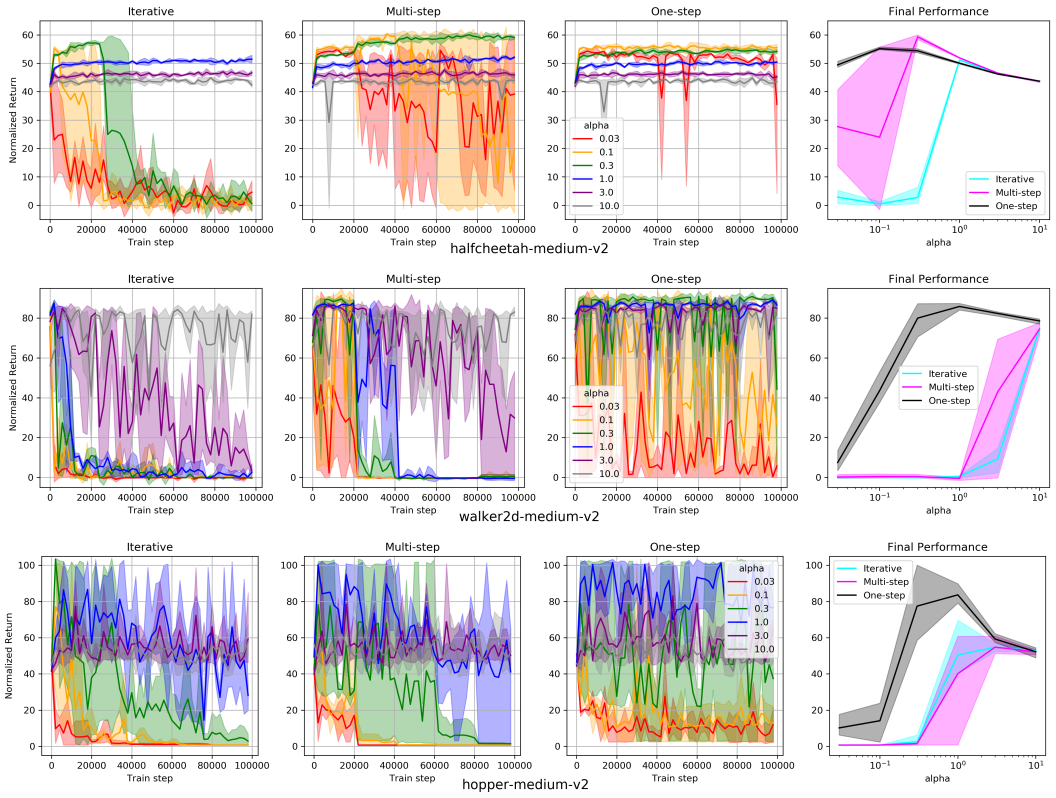


Figure A.2: Learning curves on the medium datasets.

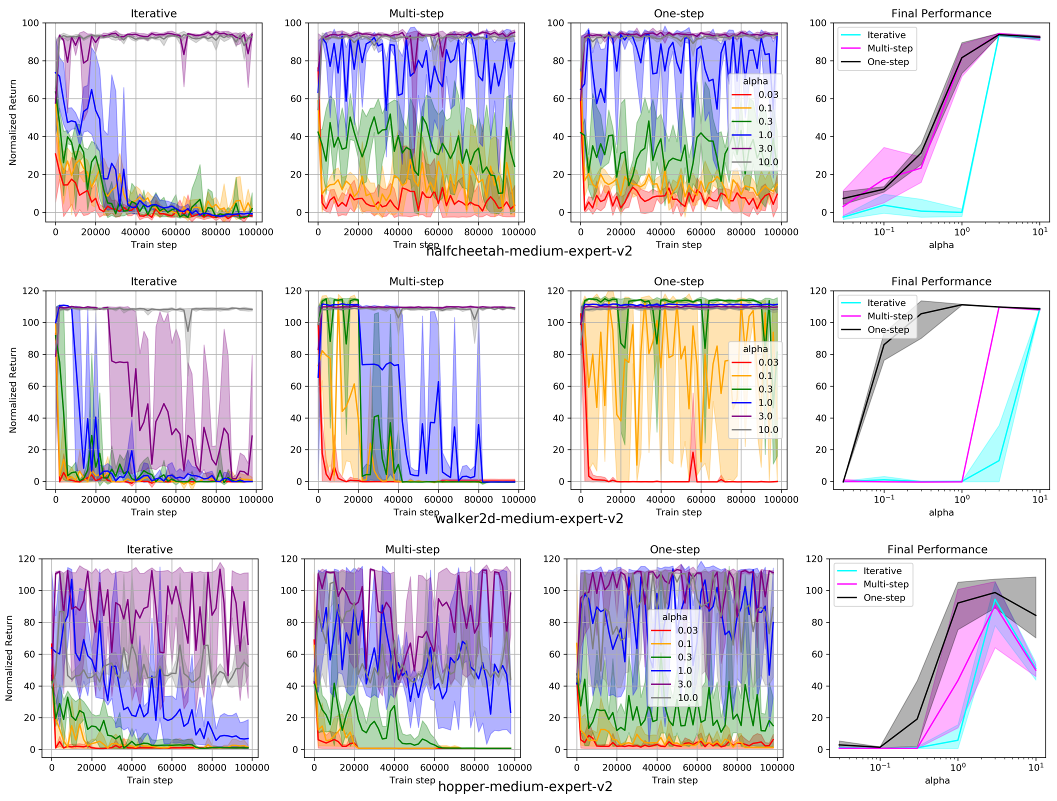


Figure A.3: Learning curves on the medium-expert datasets.

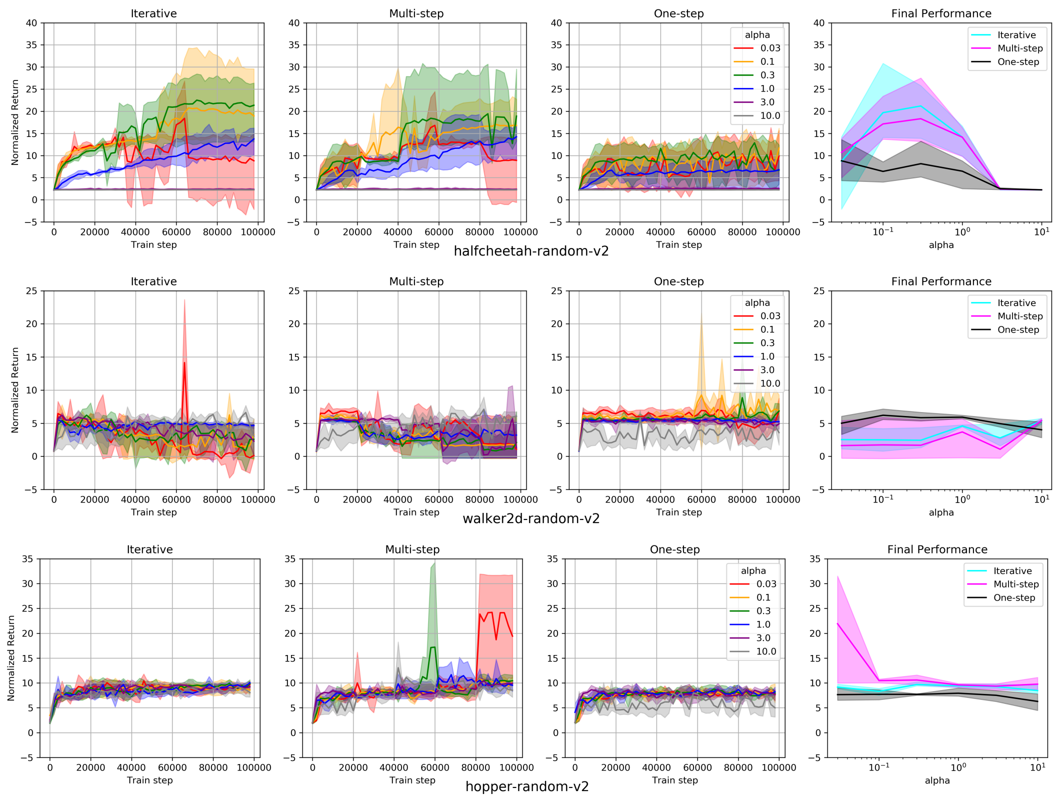


Figure A.4: Learning curves on the random datasets.

APPENDIX B

Appendix for Return Conditioned Supervised Learning

B.1 Extended experimental results

Here we present extended versions of the D4RL experiments. We use the same setup as in Section 3.6, but run each of the algorithms on three different datasets in each environment. Explicitly we show results on ANTMAZE-UMAZE, ANTMAZE-MEDIUM-PLAY, and ANTMAZE-LARGE-PLAY in Figure B.1. Then we show results on HALFCHEETAH-MEDIUM, HALFCHEETAH-MEDIUM-REPLAY, and HALFCHEETAH-MEDIUM-EXPERT in Figure B.2. Finally we show results on PEN-HUMAN, PEN-CLONED, and PEN-EXPERT in Figure B.3.

These experiments corroborate the story from the main text. Without return coverage (as in the larger antmaze tasks), RCSL can fail dramatically. But in the case with return coverage but poor state coverage (as in the pen human dataset that only has 25 trajectories), RCSL can beat DP. However we see that with larger datasets that yield more coverage, DP recovers it's performance (as in pen expert which has 5000 trajectories, or 200x the amount of data as in the human dataset).

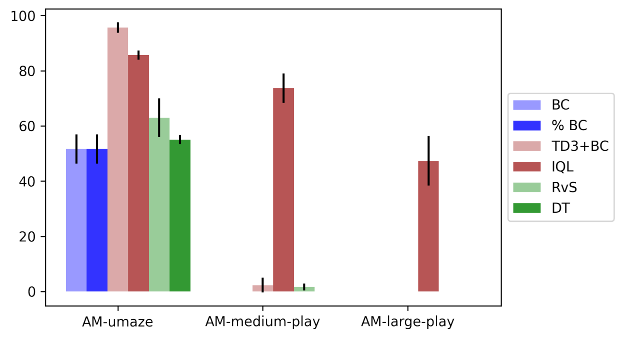


Figure B.1: Experimental results on antmaze datasets.

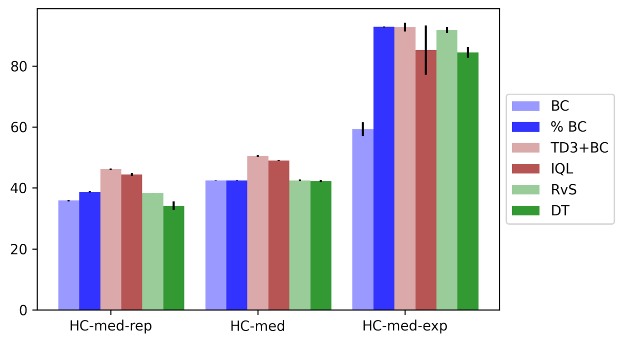


Figure B.2: Experimental results on halfcheetah datasets.

B.2 Trajectory stitching

B.2.1 Theory

A common goal from the offline RL literature is to be able to stitch together previously collected trajectories to outperform the behavior policy. This is in general not possible with RCSL. The main issue here is that RCSL is using *trajectory level* information during training, which precludes combining information across trajectories. In this example we show that even with infinite data, when attempting to combine two datastreams using standard approaches to conditioning RCSL can fail to recover the optimal policy.

Consider the MDP illustrated in Figure B.4 with three states s_0, s_1, s_2 and horizon $H = 2$. All transitions and rewards are deterministic as shown. We consider the case where data has been

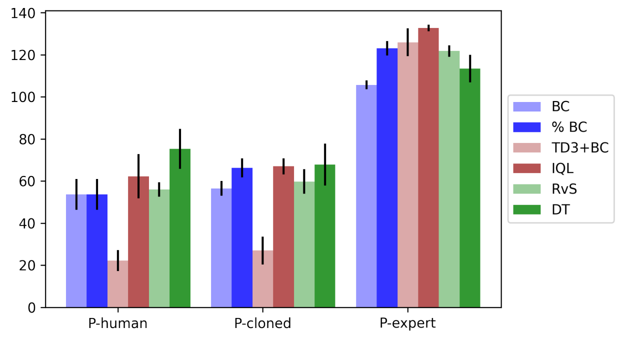


Figure B.3: Experimental results on pen datasets.

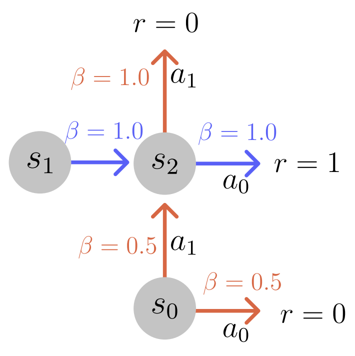


Figure B.4: An example where RCSL fails to stitch trajectories.

collected by two different processes. One process (illustrated in red) consists of episodes that always start at s_0 and chooses the first action uniformly but chooses the bad action a_0 deterministically from s_2. The other process (illustrated in blue) consists of trajectories that always start at s_1 and deterministically go towards the good action, receiving reward of 1. We will consider what happens to RCSL at test time when initialized at s_0.

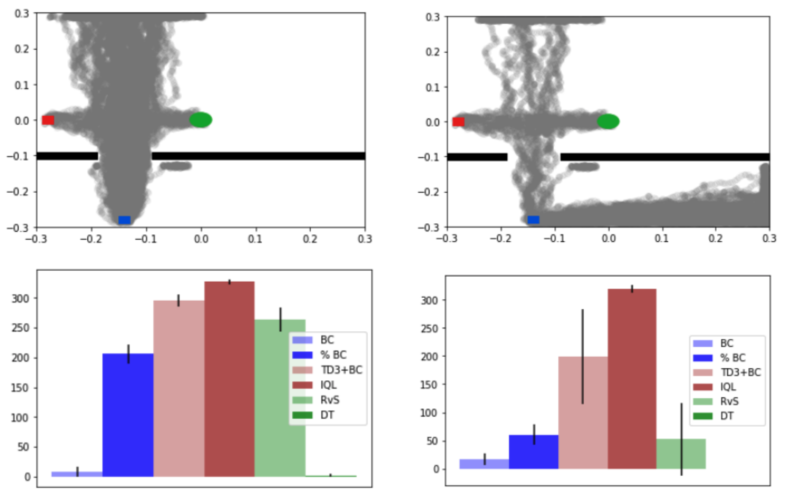
The data does not contain any trajectories that begin in s_0 and select a_1 to transition to s_2 followed by a_1, which is the optimal decision. But, the data does have enough information to stitch together the optimal trajectory from s_0, and it is clear to see that DP-based approaches would easily find the optimal policy.

For RCSL, if we condition on optimal return $g = 1$, we get that $\pi(\cdot|s_1, g = 1)$ is undefined since we only observe trajectories with $g = 0$ that originate at s_0. To get a well-defined policy, we must set $f(s_0) = 0$, but then $\pi(a_1|s_1, g = 0) = 0.5$. Thus, π will never choose the optimal path with probability larger than 0.5, for any conditioning function f. Moreover, the conditioning function that does lead to success is non-standard: $f(s_0) = 0, f(s_2) = 1$. For the standard approach to conditioning of setting the initial value and decreasing over time with observed rewards, RCSL will never achieve non-zero reward from s_0.

Note that DT-style learning where we condition on the entire history of states rather than just the current state can perform even worse since $P_{data}(a_1|s_0, a_0, s_2, g = 1) = 0$, i.e. even using the non-standard conditioning function described above will not fix things. Also, it is worth mentioning that it is possible that conditioning on the out-of-distribution return $g = 1$ from s_0 could work due to extrapolation of a neural network policy. However, as we will see in the experiments section, this does not happen empirically in controlled settings.

B.2.2 Experiments

The above example does not take into account the possibility of generalization out of distribution (i.e. when conditioning on returns that were not observed in the data). To test whether generalization could lead to stitching we construct two datasets: stitch-easy and stitch-hard. Both datasets use the same simple point-mass environment with sparse rewards as before, but now we introduce a wall into the environment to limit the paths to the goal. The stitch-easy dataset contains two types of trajectories: some beginning from the initial state region and moving upwards (with added Gaussian noise in the actions) and some beginning from the left side of the environment and moving towards the goal (with added Gaussian noise in the actions). This is "easy" since just following the behavior policy for the first half of the trajectory leads to states where the dataset indicates how to reach the goal. We also create the stitch-hard dataset which includes a third type of trajectory that begins from the initial state and goes to the right (mirroring the tabular example). This is "hard" since the dominant action from the behavior in the initial state is now to go right rather than to move towards the goal-reaching trajectories. This acts as a distraction for methods that are biased towards the behavior. Datasets and results are illustrated in Figure B.5.



(a) Stitch-easy (b) Stitch-hard

Figure B.5: Results on two datasets that require stitching.

We see that on the stitch-easy dataset RvS is able to perform almost as well as the DP algorithms and better than %BC. This indicates that it is able to follow the behavior when conditioning on an out-of-distribution return until it reaches a state where the return directs it to the goal. In contrast, DT totally fails on this task since it conditions on the entire history of the trajectory. Since the dataset only contains trajectories from the initial state that continue to the top of the board, DT always reproduces such trajectories from the initial state and does not stitch together trajectories.

In the stitch-hard dataset, we see that RvS fails as well and does not outperform %BC. This indicates that indeed, RvS can be distracted by the distractor trajectories from the initial state. The conditioning itself was not what cause the stitching in the stitch-easy dataset, but rather the learned policy simply defaults to the behavior. This can be beneficial in some problems, but prevents trajectory stitching that might allow the learned policy to dramatically outperform the behavior. TD3+BC also struggles here, likely due to some combination of instability and the BC regularization causing issues due to the distractor actions.

B.3 Proofs

B.3.1 Proof of Theorem 3.1

Proof. Let $g(s_1, a_{1:H})$ be the value of the return by rolling out the open loop sequence of actions $a_{1:H}$ under the deterministic dynamics induced by T and r. Then we can write

$$\begin{aligned}
\mathbb{E}_{s_1}[f(s_1)] - J(\pi_f) &= \mathbb{E}_{s_1}\left[\mathbb{E}_{\pi_f|s_1}[f(s_1) - g_1]\right] && \text{(B.1)}\\
&= \mathbb{E}_{s_1}\left[\mathbb{E}_{a_{1:H}\sim\pi_f|s_1}[f(s_1) - g(s_1, a_{1:H})]\right] && \text{(B.2)}\\
&\quad + \mathbb{E}_{s_1}\left[\mathbb{E}_{a_{1:H}\sim\pi_f|s_1}[g(s_1, a_{1:H}) - g_1]\right] && \text{(B.3)}\\
&\le \mathbb{E}_{s_1}\left[\mathbb{E}_{a_{1:H}\sim\pi_f|s_1}[f(s_1) - g(s_1, a_{1:H})]\right] + \epsilon H^2. && \text{(B.4)}
\end{aligned}$$

where the last step follows by bounding the magnitude of the difference between g_1 and $g(s_1, a_{1:H})$ by H and applying a union bound over the H steps in the trajectory (using the near determinism assumption), namely:

$$H \cdot \sup_{s_1} \bigcup_t P_{a_t\sim\pi_f|s_1}(r_t \neq r(s_t, a_t) \text{ or } s_{t+1} \neq T(s_t, a_t)) \le \epsilon H^2. \tag{B.5}$$

Now we consider the first term from eq. (B.4). Again bounding the magnitude of the difference by H we get that

$$\mathbb{E}_{s_1}\left[\mathbb{E}_{a_{1:H}\sim\pi_f|s_1}[f(s_1) - g(s_1, a_{1:H})]\right] \le \mathbb{E}_{s_1}\int_{a_{1:H}} P_{\pi_f}(a_{1:H}|s_1)\mathbb{1}[g(s_1, a_{1:H}) \neq f(s_1)]H \tag{B.6}$$

To bound this term, we will more carefully consider what happens under the distribution P_{π_f}. To simplify notation, let $\bar{s}_t = T(s_1, a_{1:t-1})$ be the result of following the deterministic dynamics defined by T up until step t. Expanding it out, applying the near determinism, the consistency of f, the coverage assumption, canceling some terms, and then inducting we see that:

$$P_{\pi_f}(a_{1:H}|s_1) = \pi_f(a_1|s_1) \int_{s_2} P(s_2|s_1,a_1) P_{\pi_f}(a_{2:H}|s_1,s_2) \tag{B.7}$$

$$\leq \pi_f(a_1|s_1) P_{\pi_f}(a_{2:H}|s_1,\bar{s}_2) + \epsilon \tag{B.8}$$

$$= \beta(a_1|s_1) \frac{P_\beta(g_1 = f(s_1)|s_1,a_1)}{P_\beta(g_1 = f(s_1)|s_1)} P_{\pi_f}(a_{2:H}|s_1,\bar{s}_2) + \epsilon \tag{B.9}$$

$$\leq \beta(a_1|s_1) \frac{\epsilon + P_\beta(g_1 - r(s_1,a_1) = f(s_1) - r(s_1,a_1)|s_1,a_1,\bar{s}_2)}{P_\beta(g_1 = f(s_1)|s_1)} P_{\pi_f}(a_{2:H}|s_1,\bar{s}_2) + \epsilon \tag{B.10}$$

$$= \beta(a_1|s_1) \frac{\epsilon + P_\beta(g_2 = f(\bar{s}_2)|\bar{s}_2)}{P_\beta(g_1 = f(s_1)|s_1)} P_{\pi_f}(a_{2:H}|s_1,\bar{s}_2) + \epsilon \tag{B.11}$$

$$\leq \beta(a_1|s_1) \frac{P_\beta(g_2 = f(\bar{s}_2)|\bar{s}_2)}{P_\beta(g_1 = f(s_1)|s_1)} P_{\pi_f}(a_{2:H}|s_1,\bar{s}_2) + \epsilon\left(\frac{1}{\alpha_f} + 1\right) \tag{B.12}$$

$$\leq \beta(a_1|s_1)\beta(a_2|\bar{s}_2) \frac{\cancel{P_\beta(g_2 = f(\bar{s}_2)|\bar{s}_2)}}{P_\beta(g_1 = f(s_1)|s_1)} \cdot \frac{P_\beta(g_2 = f(\bar{s}_2)|\bar{s}_2,a_2)}{\cancel{P_\beta(g_2 = f(\bar{s}_2)|\bar{s}_2)}} P_{\pi_f}(a_{3:H}|s_1,\bar{s}_3)) \tag{B.13}$$

$$+ 2\epsilon\left(\frac{1}{\alpha_f} + 1\right) \tag{B.14}$$

$$\leq \prod_{t=1}^{H} \beta(a_t|\bar{s}_t) \frac{P_\beta(g_H = f(\bar{s}_H)|\bar{s}_H,a_h)}{P_\beta(g_1 = f(s_1)|s_1)} + H\epsilon\left(\frac{1}{\alpha_f} + 1\right) \tag{B.15}$$

$$= \prod_{t=1}^{H} \beta(a_t|\bar{s}_t) \frac{\mathbb{1}[g(s_1,a_{1:H}) = f(s_1)]}{P_\beta(g_1 = f(s_1)|s_1)} + H\epsilon\left(\frac{1}{\alpha_f} + 1\right) \tag{B.16}$$

where the last step follows from the determinism of the trajectory that determines $\bar{s}_H$ and the consistency of f. Plugging this back into eq (B.6) and noticing that the two indicator functions can never both be 1, we get that:

$$\mathop{\mathbb{E}}_{s_1}\left[\mathop{\mathbb{E}}_{a_{1:H}\sim\pi_f|s_1}[f(s_1) - g(s_1,a_{1:H})]\right] \leq H^2\epsilon\left(\frac{1}{\alpha_f} + 1\right) \tag{B.17}$$

Plugging this back into eq (B.4) yields the result. □

B.3.2 Proof of Corollary 3.2

Proof. We need to define a function f so that $\mathbb{E}[f(s_1)]$ is approximately $J(\pi^*)$. To do this, note that there exists a deterministic optimal policy π^*, and since the environment dynamics are nearly deterministic we can set $f(s_1)$ to be the return of π^* under the deterministic dynamics. To do this, let $T^{\pi^*}(s_1, t)$ represent the state reached by running π^* from s_1 for t steps under the deterministic dynamics defined by T. Then:

$$f(s_1) = \sum_{t=1}^{H} r(T^{\pi^*}(s_1, t), \pi^*(T^{\pi^*}(s_1, t))) \tag{B.18}$$

Now we have as in the proof of Theorem 3.1 that the probability that $g \neq f(s)$ is bounded by ϵH, so that

$$\mathop{\mathbb{E}}_{s_1}[f(s_1)] - J(\pi^*) = \mathop{\mathbb{E}}_{s_1}[\mathop{\mathbb{E}}_{g\sim\pi^*|s_1}[f(s_1) - g]] \leq \mathop{\mathbb{E}}_{s_1}[P_{\pi^*}(g \neq f(s_1)|s_1) \cdot H] \leq \epsilon H^2 \tag{B.19}$$

Combining this with Theorem 3.1 yields the result. □

B.3.3 Proof of Theorem 3.4

First we prove the following Lemma. This can be seen as a finite-horizon analog to results from Achiam et al. [2017].

Lemma B.1. *Let d_π refer to the marginal distribution of P_π over states only. For any two policies π, π' we have:*

$$\|d_\pi - d_{\pi'}\|_1 \leq 2H \cdot \mathop{\mathbb{E}}_{s\sim d_\pi}[TV(\pi(\cdot|s)\|\hat{\pi}'(\cdot|s))] \tag{B.20}$$

Proof. First we will define a few useful objects. Let $d_\pi^h(s) = P_\pi(s_h = s)$. Let $\Delta_h = \|d_\pi^h(s) - d_{\pi'}^h(s)\|_1$.

Let $\delta_h = 2\,\mathbb{E}_{s\sim d_\pi^h}[TV(\pi(\cdot|s)\|\hat{\pi}'(\cdot|s))]$.

Now we claim that $\Delta_h \leq \delta_{h-1} + \Delta_{h-1}$ for $h > 1$ and $\Delta_1 = 0$.

To see this, consider some fixed h. Note that $d_\pi^h(s) = \int_{s'} d_\pi^{h-1}(s') \int_{a'} \pi(a'|s') P(s|s', a')$. Then expanding the definitions and adding and subtracting we see that

$$\Delta_h = \int_s |d_\pi^h(s) - d_{\pi'}^h(s)| \tag{B.21}$$

$$\leq \int_s \left| \int_{s'} d_\pi^{h-1}(s') \int_{a'} (\pi(a'|s') - \pi'(a'|s'))P(s|s', a') \right| \tag{B.22}$$

$$+ \int_s \left| \int_{s'} (d_\pi^{h-1}(s') - d_{\pi'}^{h-1}(s')) \int_{a'} \pi'(a'|s')P(s|s', a') \right| \tag{B.23}$$

$$\leq 2 \underset{s\sim d_\pi^{h-1}}{\mathbb{E}}[TV(\pi(\cdot|s)\|\hat{\pi}'(\cdot|s))] + \|d_\pi^{h-1} - d_{\pi'}^{h-1}\|_1 = \delta_{h-1} + \Delta_{h-1}. \tag{B.24}$$

Now applying the claim and the definition of d_π we get that

$$\|d_\pi - d_{\pi'}\|_1 \leq \frac{1}{H}\sum_{h=1}^{H} \Delta_h \leq \frac{1}{H}\sum_{h=1}^{H}\sum_{j=1}^{h-1} \delta_j \leq H\frac{1}{H}\sum_{h=1}^{H} \delta_h = 2H \cdot \underset{s\sim d_\pi}{\mathbb{E}}[TV(\pi(\cdot|s)\|\hat{\pi}'(\cdot|s))]. \tag{B.25}$$

□

Now we can prove the Theorem.

Proof. Applying the definition of J and Lemma B.1, we get

$$J(\pi_f) - J(\hat{\pi}_f) = H(\underset{P_{\pi_f}}{\mathbb{E}}[r(s,a)] - \underset{P_{\hat{\pi}_f}}{\mathbb{E}}[r(s,a)]) \tag{B.26}$$

$$\leq H\|d_{\pi_f} - d_{\hat{\pi}_f}\|_1 \tag{B.27}$$

$$\leq 2 \cdot \underset{s\sim d_{\pi_f}}{\mathbb{E}}[TV(\pi_f(\cdot|s)\|\hat{\pi}_f(\cdot|s))]H^2 \tag{B.28}$$

Expanding definitions, using the multiply and divide trick, and applying the assumptions:

$$\begin{aligned}
2 \cdot \mathop{\mathbb{E}}_{s\sim d_{\pi_f}}[TV(\pi_f(\cdot|s)\|\hat{\pi}_f(\cdot|s))] &= \mathop{\mathbb{E}}_{s\sim d_{\pi_f}}\left[\int_a |P_\beta(a|s,f(s)) - \hat{\pi}(a|s,f(s))|\right] && \text{(B.29)}\\
&= \mathop{\mathbb{E}}_{s\sim d_{\pi_f}}\left[\frac{P_\beta(f(s)|s)}{P_\beta(f(s)|s)}\int_a |P_\beta(a|s,f(s)) - \hat{\pi}(a|s,f(s))|\right] && \text{(B.30)}\\
&\leq \frac{C_f}{\alpha_f}\mathop{\mathbb{E}}_{s\sim d_\beta}\left[P_\beta(f(s)|s)\int_a |P_\beta(a|s,f(s)) - \hat{\pi}(a|s,f(s))|\right] && \text{(B.31)}\\
&\leq \frac{C_f}{\alpha_f}\mathop{\mathbb{E}}_{s\sim d_\beta}\left[\int_g P_\beta(g|s)\int_a |P_\beta(a|s,g) - \hat{\pi}(a|s,g)|\right] && \text{(B.32)}\\
&= 2\frac{C_f}{\alpha_f}\mathop{\mathbb{E}}_{s\sim d_{\pi_f}, g\sim P_\beta|s}[TV(P_\beta(\cdot|s,g)\|\hat{\pi}(\cdot|s,g))] && \text{(B.33)}\\
&\leq \frac{C_f}{\alpha_f}\sqrt{2L(\hat{\pi})} && \text{(B.34)}
\end{aligned}$$

where the last step comes from Pinsker's inequality. Combining with the above bound on the difference in expected values yields the result. □

B.3.4 Proof of Corollary 3.5

Proof. We may write $L(\pi) = \bar{L}(\pi) - H_\beta$, where $H_\beta = -\mathbb{E}_{(s,a,g)\sim P_\beta}[\log P_\beta(a|s,g)]$ and

$$\bar{L}(\pi) := -\mathop{\mathbb{E}}_{(s,a,g)\sim P_\beta}[\log \pi(a|s,g)]$$

is the cross-entropy loss. Denoting $\pi^\dagger \in \arg\min_{\pi\in\Pi} L(\pi)$, we have

$$L(\hat{\pi}) = L(\hat{\pi}) - L(\pi^\dagger) + L(\pi^\dagger) \leq \bar{L}(\hat{\pi}) - \bar{L}(\pi^\dagger) + \epsilon_{approx}.$$

Denoting $\hat{L}$ the empirical cross-entropy loss that is minimized by $\hat{\pi}$, we may further decompose

$$\begin{aligned}\bar{L}(\hat{\pi}) - \bar{L}(\pi^\dagger) &= \bar{L}(\hat{\pi}) - \hat{L}(\hat{\pi}) + \hat{L}(\hat{\pi}) - \hat{L}(\pi^\dagger) + \hat{L}(\pi^\dagger) - \bar{L}(\pi^\dagger) \\ &\leq 2 \sup_{\pi \in \Pi} |\bar{L}(\pi) - \hat{L}(\pi)|\end{aligned}$$

Under the assumptions on bounded loss differences, we may bound this, e.g., using McDiarmid's inequality and a union bound on Π to obtain the final result. □

B.3.5 Top-% BC

Theorem B.2 (Alignment with respect to quantile). *Let g_ρ be the $1-\rho$ quantile of the return distribution induced by β over all initial states. Let $\pi_\rho = P_\beta(a|s, g \geq g_\rho)$. Assume the following:*

1. *Coverage: $P_\beta(s_1|g \geq g_\rho) \geq \alpha_\rho$ for all initial states s_1.*
2. *Near determinism: $P(r \neq r(s,a) \text{ or } s' \neq T(s,a)|s,a) \leq \epsilon$ at all s, a for some functions T and r. Note that this does not constrain the stochasticity of the initial state at all.*

Then

$$g_\rho - J(\pi_\rho) \leq \epsilon \left(\frac{1}{\alpha_\rho} + 2 \right) H^2. \tag{B.35}$$

Proof. The proof essentially follows the same argument as Theorem 3.1 with $f(s_1)$ replaced by g_ρ. The main difference comes from the fact that

$$\pi_\rho(a|s) = P_\beta(a|s, g \geq g_\rho) = \beta(a|s) \frac{P_\beta(g \geq g_\rho|s, a)}{P_\beta(g \geq g_\rho|s)} \tag{B.36}$$

Explicitly, we have similar to before that:

$$g_\rho - J(\pi_\rho) = \mathop{\mathbb{E}}_{s_1}[\mathop{\mathbb{E}}_{\pi_\rho|s_1}[g_\rho - g_1]] \tag{B.37}$$

$$\leq \mathop{\mathbb{E}}_{s_1} \mathop{\mathbb{E}}_{a_{1:H}\sim\pi_f|s_1}[g_\rho - g(s_1, a_{1:H})] + \epsilon H^2. \tag{B.38}$$

$$\leq \mathop{\mathbb{E}}_{s_1} \mathop{\mathbb{E}}_{a_{1:H}\sim\pi_f|s_1}[\mathbb{1}[g(s_1, a_{1:H}) < g_\rho]] \cdot H + \epsilon H^2. \tag{B.39}$$

We now define $\bar{s}_t = T(s_1, a_{1:t-1})$ to be the state at step t under the determinisitic dynamics and similarly $\bar{r}_t = r(\bar{s}_t, a_t)$ the reward under deterministic dynamics. Then again mirroring the proof above, we have that

$$P_{\pi_\rho}(a_{1:H}|s_1) \leq \pi_\rho(a_1|s_1)P_{\pi_\rho}(a_{2:H}|s_1, \bar{s}_2, \bar{r}_1) + \epsilon \tag{B.40}$$

$$= \beta(a_1|s_1)\frac{P_\beta(g_1 \geq g_\rho|s_1, a_1)}{P_\beta(g_1 \geq g_\rho|s_1)}P_{\pi_\rho}(a_{2:H}|s_1, \bar{s}_2, \bar{r}_1) + \epsilon \tag{B.41}$$

$$\leq \beta(a_1|s_1)\frac{\epsilon + P_\beta(g_1 \geq g_\rho|\bar{s}_2, \bar{r}_1, a_1)}{P_\beta(g_1 \geq g_\rho|s_1)}P_{\pi_\rho}(a_{2:H}|s_1, \bar{s}_2, \bar{r}_1) + \epsilon \tag{B.42}$$

$$\leq \beta(a_1|s_1)\frac{P_\beta(g_1 \geq g_\rho|s_1, a_1)}{P_\beta(g_1 \geq g_\rho|s_1)}P_{\pi_\rho}(a_{2:H}|s_1, \bar{s}_2, \bar{r}_1) + \epsilon \tag{B.43}$$

$$\leq \beta(a_1|s_1)\frac{P_\beta(g_1 \geq g_\rho|\bar{s}_2, \bar{r}_1, a_1)}{P_\beta(g_1 \geq g_\rho|s_1)}P_{\pi_\rho}(a_{2:H}|s_1, \bar{s}_2, \bar{r}_1) + \epsilon\left(\frac{1}{\alpha_\rho} + 1\right) \tag{B.44}$$

$$\leq \beta(a_1|s_1)\beta(a_2|\bar{s}_2)\frac{\cancel{P_\beta(g_1 \geq g_\rho|\bar{s}_2, \bar{r}_1)}}{P_\beta(g_1 \geq g_\rho|s_1)}\frac{P_\beta(g_1 \geq g_\rho|\bar{s}_2, \bar{r}_1, a_2)}{\cancel{P_\beta(g_1 \geq g_\rho|\bar{s}_2, \bar{r}_1)}}P_{\pi_\rho}(a_{3:H}|s_1, \bar{s}_3, \bar{r}_{1:2}) \tag{B.45}$$

$$+ 2\epsilon\left(\frac{1}{\alpha_\rho} + 1\right) \tag{B.46}$$

$$\leq \prod_{t=1}^{H}\beta(a_t|\bar{s}_t)\frac{P_\beta(g_1 \geq g_\rho|\bar{s}_H, \bar{r}_{1:H})}{P_\beta(g_1 \geq g_\rho|s_1)} + H\epsilon\left(\frac{1}{\alpha_\rho} + 1\right) \tag{B.47}$$

$$= \prod_{t=1}^{H}\beta(a_t|\bar{s}_t)\frac{\mathbb{1}[g(s_1, a_{1:H}) \geq g_\rho]}{P_\beta(g_1 \geq g_\rho|s_1)} + H\epsilon\left(\frac{1}{\alpha_\rho} + 1\right) \tag{B.48}$$

Plugging this into Equation B.39 we get the result. □

Theorem B.3 (Reduction of %BC to SL). *Let g_ρ be the $1 - \rho$ percentile of the return distribution* *induced by β. Let $\pi_\rho = P_\beta(a|s, g \geq g_\rho)$. Assume*

1. *Bounded mismatch: $\frac{P_{\pi_\rho}(s)}{P_\beta(s|g\geq g_\rho)} \leq C_\rho$ for all s.*

Define the expected loss as $L_\rho(\hat{\pi}) = \mathbb{E}_{s\sim P_\beta|g\geq g_\rho}[KL(\pi_\rho(\cdot|s)\|\hat{\pi}(\cdot|s))]$. Then we have that

$$J(\pi_\rho) - J(\hat{\pi}) \leq C_\rho H^2\sqrt{2L_\rho(\hat{\pi})}. \tag{B.49}$$

Proof. Recall that d_π refers to the marginal distribution of P_π over states only. Applying the definition of J and Lemma B.1, we get

$$\begin{aligned} J(\pi_\rho) - J(\hat{\pi}) &= H(\mathop{\mathbb{E}}_{P_{\pi_\rho}}[r(s,a)] - \mathop{\mathbb{E}}_{P_{\hat{\pi}}}[r(s,a)]) && \text{(B.50)}\\ &\leq H\|d_{\pi_\rho} - d_{\hat{\pi}}\|_1 && \text{(B.51)}\\ &\leq 2\cdot \mathop{\mathbb{E}}_{s\sim d_{\pi_\rho}}[TV(\pi_\rho(\cdot|s)\|\hat{\pi}(\cdot|s))]H^2 && \text{(B.52)} \end{aligned}$$

Expanding definitions, using the multiply and divide trick, and applying the assumptions:

$$\begin{aligned} 2\cdot \mathop{\mathbb{E}}_{s\sim d_{\pi_\rho}}[TV(\pi_\rho(\cdot|s)\|\hat{\pi}(\cdot|s))] &\leq C_\rho \cdot 2 \mathop{\mathbb{E}}_{s\sim P_\beta(\cdot|g\geq g_\rho)}[TV(\pi_\rho(\cdot|s)\|\hat{\pi}(\cdot|s))] && \text{(B.53)}\\ &\leq C_\rho\sqrt{2L(\hat{\pi})} && \text{(B.54)} \end{aligned}$$

where the last step comes from Pinsker's inequality. Combining with the above bound on the difference in expected values yields the result. □

Corollary B.4 (Sample complexity for %BC)**.** *To get finite data guarantees, add to the above assumptions the assumptions that (1) the policy class Π is finite, (2) $|\log\pi(a|s) - \log\pi(a'|s')| \leq c$ for any (a, s, a', s') and all $\pi \in \Pi$, and (3) the approximation error of Π is bounded by ϵ_{approx}, i.e.*

$\min_{\pi \in \Pi} L_\rho(\pi) \leq \epsilon_{approx}$. *Then with probability at least* $1-\delta$,

$$J(\pi_\rho) - J(\hat{\pi}) \leq O\left(C_\rho H^2 \left(\sqrt{c}\left(\frac{\log|\Pi|/\delta}{(1-\rho)N}\right)^{1/4} + \sqrt{\epsilon_{approx}}\right) + \frac{\epsilon}{\alpha_\rho} H^2\right). \tag{B.55}$$

B.4 Experimental details

Data. Data for point-mass tasks was sampled from the scripted policies described in the text. We sampled 100 trajectories of length 400 for each dataset, unless otherwise indicated. Data for the benchmark experiments was taken directly from the D4RL benchmark [Fu et al. 2020].

Hyperparameters. Below we list all of the hyperparameters used across the various agorithms. We train each algorithm on 90% of the trajectories in the dataset, using the remaining 10% as validation. All algorithms are trained with the Adam optimizer [Kingma and Ba 2014]. We evaluate each algorithm for 100 episodes in the environment per seed and hyperparameter configuration and report the best performance for each algorithm for it's relevant hyperparameter (All algorithms were tuned across 3 values of the hyperparameter except for DT on pointmass where we tried more values, but still got poor results). Error bars are reported across seeds, as explained in the text.

Table B.1: Shared hyperparameters for all non-DT algorithms

Hyperparameter	Value
Training steps	$5e5$
Batch size	256
MLP width	256
MLP depth	2

Compute. All experiments were run on CPU on an internal cluster. Each of the non-DT algorithms takes less than 1 hour per run (i.e. set of hyperparameters and seed) and the DT algorithm

Table B.2: Algorithm-specific hyperparameters for all non-DT algorithms

Algorithm	Hyperparameter	Value(s)
%BC	fraction ρ	[0.02, 0.10, 0.5]
	learning rate	1e-3
RvS	fraction of max return for conditioning	[0.8, 1.0, 1.2]
	learning rate	1e-3
TD3+BC	α	[1.0, 2.5, 5.0]
	learning rate (actor and critic)	3e-4
	discount	0.99
	τ for target EWMA	0.005
	target update period	2
IQL	expectile	[0.5, 0.7, 0.9]
	learning rate (actor, value, and critic)	3e-4
	discount	0.99
	τ for target EWMA	0.005
	temperature	10.0

Table B.3: Hypereparameters for DT (exactly as in [Chen et al. 2021b])

Hyperparameter	Value
Training steps	$1e5$
Batch size	64
Learning rate	1e-4
Weight decay	1e-4
K	20
Embed dimension	128
Layers	3
Heads	1
Dropout	0.1

takes 5-10 hours per run.

Asset licenses. For completeness, we also report the licenses of the assets that we used in the paper: JAX [Bradbury et al. 2018]: Apache-2.0, Flax [Heek et al. 2020]: Apache-2.0, jaxrl [Kostrikov 2021]: MIT, Decision Transformer [Chen et al. 2021b]: MIT, Deepmind control suite [Tassa et al. 2018]: Apache-2.0, mujoco [Todorov et al. 2012a]: Apache-2.0, D4RL [Fu et al. 2020]: Apache-2.0.

Code. Please see https://github.com/davidbrandfonbrener/rcsl-paper.

Table B.4: Environment-specific reward targets for DT

Environment	Values
Point-mass	[300, 200, 100, 50, 10, 0]
Antmaze	[1.0, 0.75, 0.5]
Half-cheetah	[12000, 9000, 6000]
Pen	[3000, 2000, 1000]

APPENDIX C

Appendix for Exploratory Data

C.1 Hyper-parameters

In this section we comprehensively describe configuration of both unsupervised RL and offline RL algorithms.

C.1.1 Unsupervised RL Hyper-parameters

We adhere closely to the parameter settings from URLB [Laskin et al. 2021]. We list the common hyper-parameters in Table C.1 and per-algorithm hyper-parameters in Table C.2.

Table C.1: Common hyper-parameters for unsupervised RL algorithms.

Common hyper-parameter	Value
Replay buffer capacity	10^6
Seed frames	4000
Mini-batch size	1024
Discount (γ)	0.99
Optimizer	Adam
Learning rate	10^{-4}
Agent update frequency	2
Critic target EMA rate (τ_Q)	0.01
Hidden dim.	1024
Exploration stddev clip	0.3
Exploration stddev value	0.2

Table C.2: Individual hyper-parameters for unsupervised RL algorithms.

Random hyper-parameter	Value
Policy distribution	uniform
ICM hyper-parameter	**Value**
Reward transformation	$\log(r + 1.0)$
Forward net arch.	$(\lvert\mathcal{O}\rvert + \lvert\mathcal{A}\rvert) \rightarrow 1024 \rightarrow 1024 \rightarrow \lvert\mathcal{O}\rvert$ ReLU MLP
Inverse net arch.	$(2 \times \lvert\mathcal{O}\rvert) \rightarrow 1024 \rightarrow 1024 \rightarrow \lvert\mathcal{A}\rvert$ ReLU MLP
Disagreement hyper-parameter	**Value**
Ensemble size	5
Forward net arch:	$(\lvert\mathcal{O}\rvert + \lvert\mathcal{A}\rvert) \rightarrow 1024 \rightarrow 1024 \rightarrow \lvert\mathcal{O}\rvert$ ReLU MLP
RND hyper-parameter	**Value**
Predictor & target net arch.	$\lvert\mathcal{O}\rvert \rightarrow 1024 \rightarrow 1024 \rightarrow 512$ ReLU MLP
Normalized observation clipping	5
APT hyper-parameter	**Value**
Reward transformation	$\log(r + 1.0)$
Forward net arch.	$(512 + \lvert\mathcal{A}\rvert) \rightarrow 1024 \rightarrow 512$ ReLU MLP
Inverse net arch.	$(2 \times 512) \rightarrow 1024 \rightarrow \lvert\mathcal{A}\rvert$ ReLU MLP
k in NN	12
Avg top k in NN	True
Proto hyper-parameter	**Value**
Predictor dim.	128
Projector dim.	512
Number of prototypes	512
Softmax temperature	0.1
k in NN	3
Number of candidates per prototype	4
Encoder target EMA rate (τ_{enc})	0.05
SMM hyper-parameter	**Value**
Skill dim.	4
Skill discriminator learning rate	10^{-3}
VAE lr	10^{-2}
DIAYN hyper-parameter	**Value**
Skill dim	16
Skill sampling frequency (steps)	50
Discriminator net arch.	$512 \rightarrow 1024 \rightarrow 1024 \rightarrow 16$ ReLU MLP
APS hyper-parameter	**Value**
Reward transformation	$\log(r + 1.0)$
Successor feature dim.	10
Successor feature net arch.	$\lvert\mathcal{O}\rvert \rightarrow 1024 \rightarrow 1024 \rightarrow 10$ ReLU MLP
k in NN	12
Avg top k in NN	True
Least square batch size	4096

C.1.2 Offline RL Hyper-parameters

In Table C.3 we present a common set of hyper-parameters used in our experiments, while in Table C.4 we list individual hyper-parameters for each method. To select hyper-parameters for offline RL algorithms we started with from the original hyper-parameters presented in the corresponding papers. We then performed a grid search over the most important parameters (i.e. learning rate, α, mini-batch size, etc.) on a supervised dataset collected by TD3 [Fujimoto et al. 2018b] on Cheetah Run.

TD3+BC We swept over learning rate $[10^{-3}, 3 \cdot 10^{-4}, 10^{-4}]$, mini-batch size $[256, 512, 1024]$ and α $[0.01, 0.1, 1.0, 2.5, 5.0, 10.0, 50.0]$. We also found it useful to increase the dimension of hidden layers to 1024.

CRR We swept over the transformation function (indicator, exponent, identity), the number of samples to estimate value function $[5, 10, 20]$ as well as learning rate $[10^{-3}, 3 \cdot 10^{-4}, 10^{-4}]$ and mini-batch size $[256, 512, 1024]$.

CQL We swept over α $[0.01, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0]$, the number of sampled actions $[3, 5, 10, 20]$, learning rate $[10^{-3}, 3 \cdot 10^{-4}, 10^{-4}]$ and mini-batch size $[256, 512, 1024]$.

Table C.3: Common hyper-parameters for offline RL algorithms.

Common hyper-parameter	Value
Replay buffer capacity	10^6
Mini-batch size	1024
Discount (γ)	0.99
Optimizer	Adam
Learning rate	10^{-4}
Agent update frequency	2
Training steps	5×10^5

Table C.4: Individual hyper-parameters for offline RL algorithms.

BC hyper-parameter	Value
Num hidden layers	2
Hidden dim.	1024
TD3+BC hyper-parameter	**Value**
α	2.5
Critic target EMA rate (τ_Q)	0.01
Num hidden layers	2
Hidden dim.	1024
CRR hyper-parameter	**Value**
Num samples to estimate V	10
Critic target EMA rate (τ_Q)	0.01
Transformation	indicator
Num hidden layers	2
Hidden dim.	1024
CQL hyper-parameter	**Value**
α	0.01
Critic target EMA rate (τ_Q)	0.01
Lagrange	False
Num sampled actions	3
Num hidden layers	2
Hidden dim.	1024
TD3 hyper-parameter	**Value**
Stddev clip	0.3
Critic target EMA rate (τ_Q)	0.01
Num hidden layers	2
Hidden dim.	1024

C.2 Full Results on the PointMass Experiment

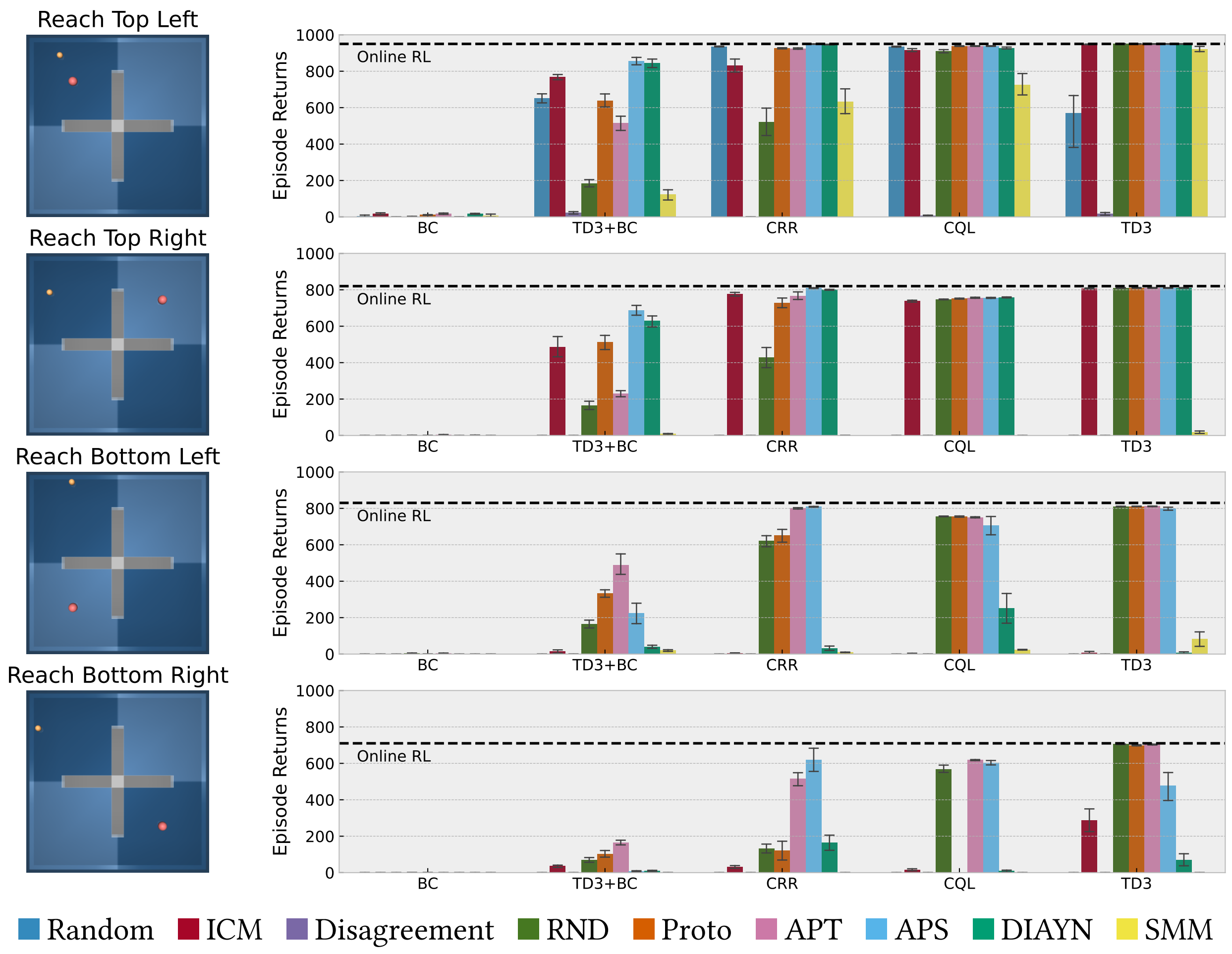


Figure C.1: Extended figures from Figure 4.2.

C.3 Full Results on Singletask Settings

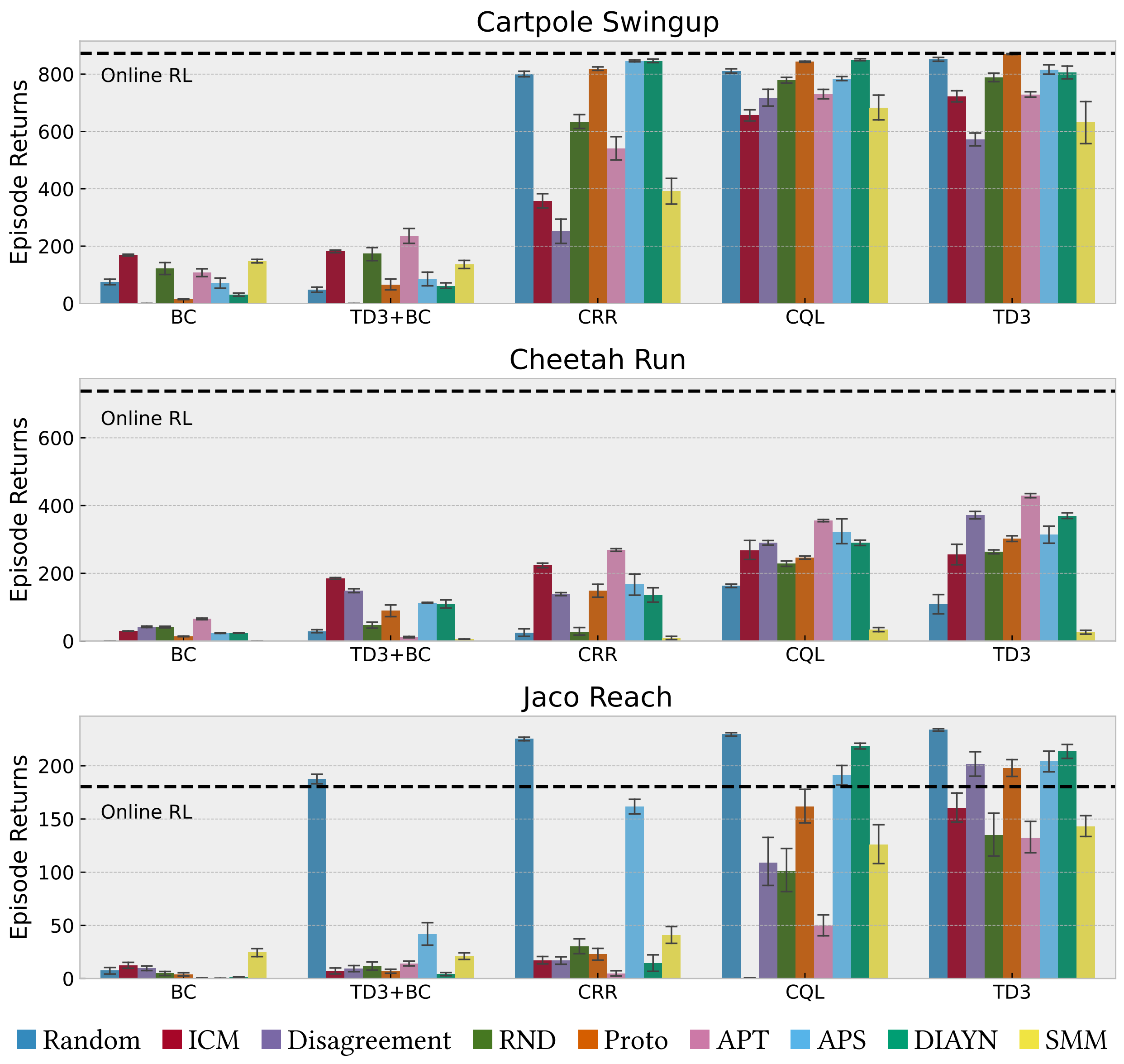


Figure C.2: Offline evaluation of unsupervised datasets on one task for each of three different domains. Here we choose four representative unsupervised exploration algorithms.

C.4 Full Results on Multitask Settings

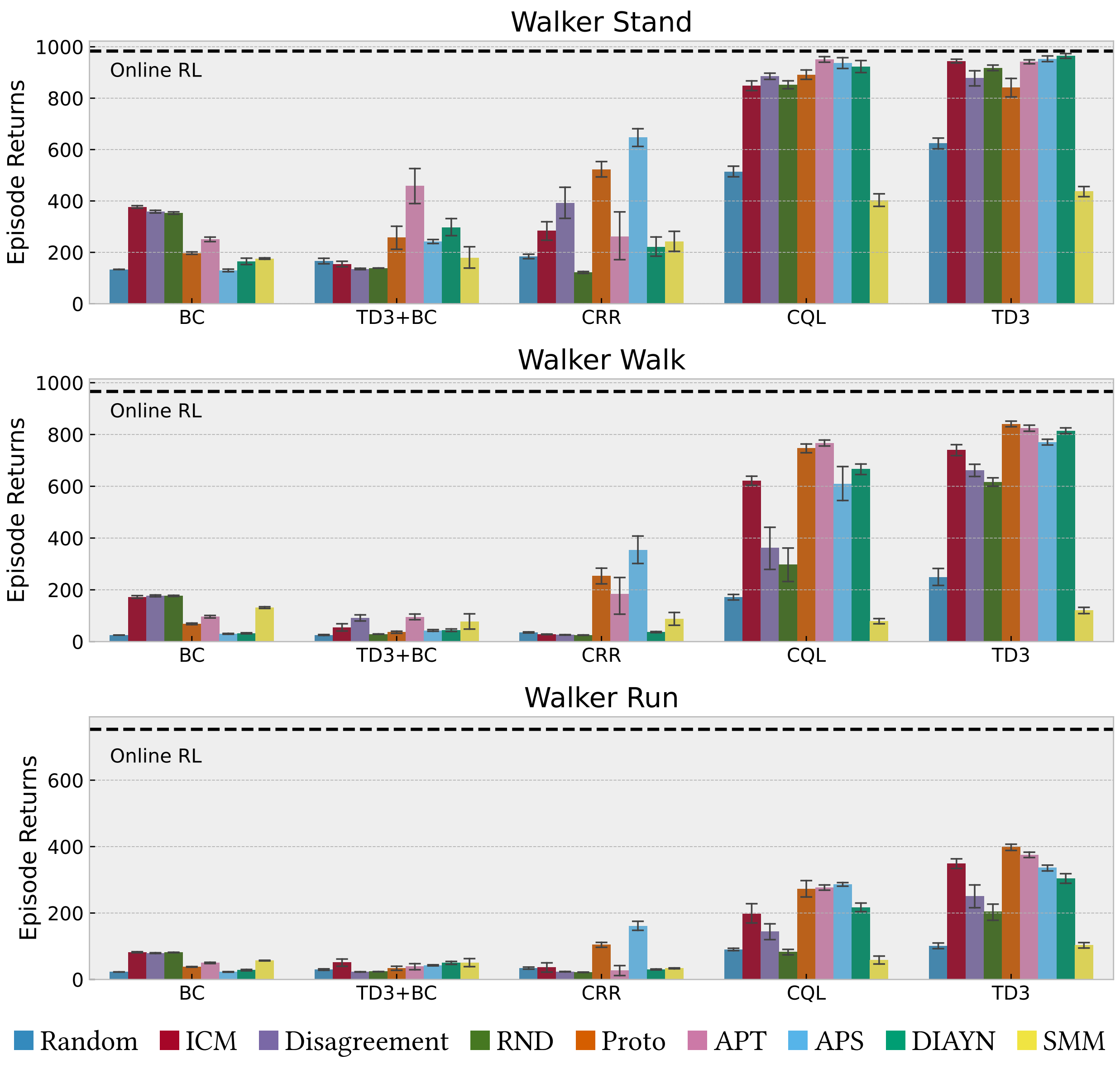


Figure C.3: Offline evaluation of datasets for the Walker environment under three different rewards (Stand, Walk, and Run). We observe that ExORL allows for data relabeling to enable multi-task offline RL.

C.5 Offline RL on Suffixes of Unsupervised Datasets

In Figure C.4 we demonstrate that it is important to use the entire replay buffers collected by ExORL, rather than only training on the later parts of this data.

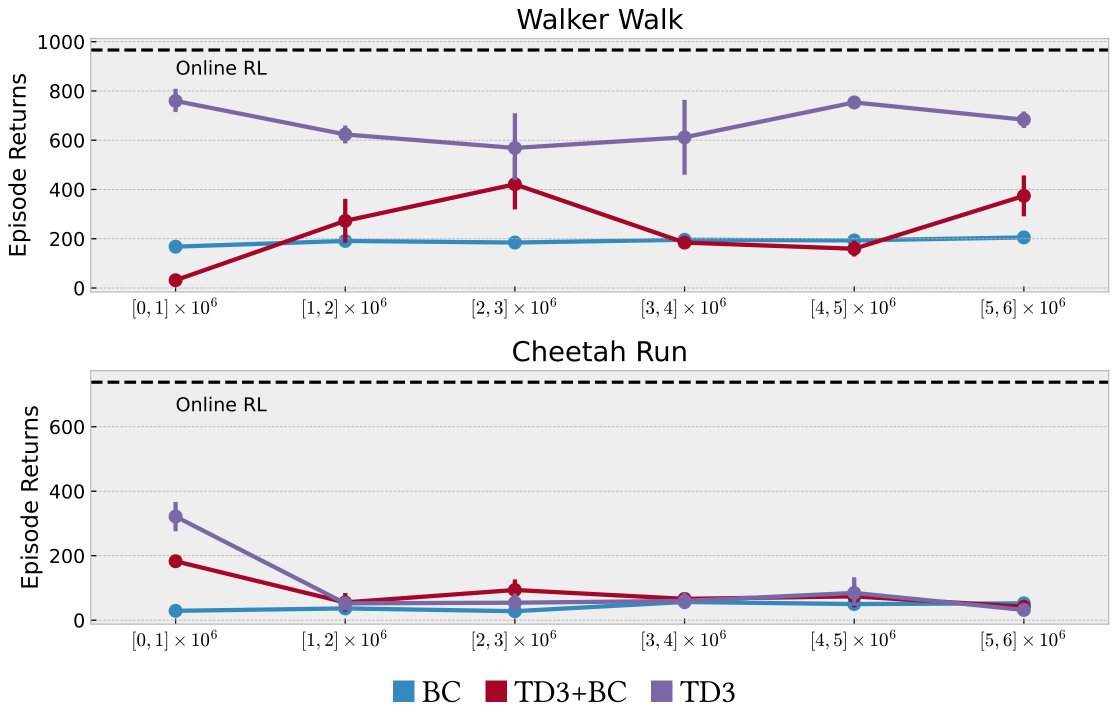


Figure C.4: Instad of training on the full replay buffer collected by ICM, we perform offline RL training on various suffixes of the replay buffer. The x-axis represents segments of the replay buffuer. We observe that it is important to keep around early transitions.

C.6 Compute Resources

ExORL is designed to be accessible to the RL research community. We provide an efficient implementation of our framework, including data collection, relabeling, and offline RL evaluation, that requires a single GPU. For local debugging experiments we used NVIDIA RTX GPUs. For large-scale runs used to generate all results in this manuscripts, we used NVIDIA Tesla V100 GPU instances. All experiments were run on internal clusters. Each offline RL algorithm trains in roughly 2 hours for 500k gradient steps on the collected datasets.

C.7 The ExORL Environments and Tasks

We provide a summary of used environments and tasks in our paper in Table C.5.

Environment	Task	Traits	$\dim(\mathcal{S})$	$\dim(\mathcal{A})$
Walker	Stand	dense, easy exp, locomotion	18	6
	Walk	dense, medium exp, locomotion	18	6
	Run	dense, hard exp locomotion	18	6
	Flip	dense, medium exp, locomotion	18	6
Cheetah	Run	dense, hard exp, locomotion	24	6
	Run Backward	dense, hard exp, hard transfer	24	6
PointMass Maze	Reach Top Left	sparse, hard exp, hard transfer	4	2
	Reach Top Right	sparse, hard exp, hard transfer	4	2
	Reach Bot Left	sparse, hard exp, hard transfer	4	2
	Reach Bot Right	sparse, hard exp, hard transfer	4	2
Jaco	Reach	sparse, hard exp, manipulation	55	6
Cartpole	Swingup	dense, easy exp	4	1

Table C.5: A detailed description of used environments and tasks from the DeepMind control suite [Tassa et al. 2018]. Note that "sparse" and "dense" refer to the reward and "exp" for exploration.

APPENDIX D

Appendix for Inverse Dynamics Pretraining

D.1 Extended related work

In this paper we focus specifically on pretraining methods that learn representations of high dimensional observations from multitask demonstration data with latent contexts for the purpose of imitation. There are many closely related problems that are studied in other papers that we did not have space to address fully in the main text that we more fully describe here. These are all very interesting and complementary lines of work, but are beyond the scope of this paper.

Perhaps the largest closely related line of work focuses on learning reward-directed representations in the context of reinforcement learning. This is a different setting than ours and methods from there are not applicable in our setting where we do not have access to rewards. Moreover, most of these methods do not consider multitask settings [Zhang et al. 2020; Gelada et al. 2019; Fu et al. 2021b; Ghosh et al. 2018; Eysenbach et al. 2022; Sodhani et al. 2021].

Another line of work seeks to learn representations of actions or sequences of actions rather than observations. This is a directly complementary line of work to the ideas presented in this paper [Ajay et al. 2020; Yang et al. 2021; Lynch et al. 2020; Whitney et al. 2019].

Another body of literature focuses on learning representations that can be transferred across domain and embodiment gaps and even trained directly from videos without access to actions at all [Oord et al. 2018; Aytar et al. 2018; Seo et al. 2022; Ma et al. 2022; Zakka et al. 2022; Ghosh et al. 2023]. In this paper, we focus on the simpler task of pretraining a representation within one MDP with consistent dynamics and access to demonstration actions, but with varied tasks. This choice allows us to make more clear comparisons between algorithms and rigorous claims about when representations will be effective, but also limits the generality of the representations that are learned.

There are a variety of new methods that rely on transformer architectures to construct interesting new pretraining objectives [Yang and Nachum 2021; Seo et al. 2023; Wu et al. 2023]. In this paper we focus on simple methods that can all use the same simple convolutional architecture operating on transition tuples to provide the most controlled comparison that we can. It is an interesting direction for future work to see how our insights in the Markovian case could be leveraged to inform sequence level models of partially observed problems.

Another line of work avoids pretraining representations directly and instead meta-learns a policy that can adapt to new tasks [Duan et al. 2017; Finn et al. 2017a,b; Yu et al. 2018; Rakelly et al. 2019; Mitchell et al. 2021]. This approach is beyond the scope of this paper which focuses on representation learning. Moreover, these meta-learning algorithms require the pretraining trajectories to be partitioned by task so that each task has multiple trajectories. Since we focus on pretraining data where we don't have access to the latent context, it is unclear how to create these meta-training datasets.

Finally, recent work has shown the promise of zero-shot generalization for multitask imitation, especially when the task identifying information is expressed in natural language to leverage advances in language models [Ding et al. 2019; Jang et al. 2022; Cui et al. 2022; Brohan et al. 2022]. This is an exciting line of work, but beyond the scope of this project where we focus on data where the context information is latent. It is an interesting direction for future work to assess precisely how much performance can be improved via extra context information to gauge whether it is worth the cost of labeling trajectories with context information.

It is an interesting direction for future work to try to better synthesize some of the findings from across this broad array of approaches to pretraining in slightly different settings.

D.2 Extended experimental results

In this section we present the experimental results that were excluded from the main text due to space constraints. In particular, Section D.2.1 presents representation analysis by predicting one representation from another, Section D.2.2 presents the per-dataset results of various sweeps over dataset size and type, Section D.2.3 presents per-dataset results for representation analysis, and Section D.2.4 presents results of an ablation over multistep dynamics.

D.2.1 Cross-representation prediction

In the main text, we evaluated representation quality by measuring accuracy of small MLPs to predict either the actions on the finetuning data or the low dimensional states on the pretraining data. Here we present a similar analysis, but now where we use small MLPs to predict the other representations themselves. This is interesting since it lets us assess which representations contain enough information and shared structure to predict the other representations. Hypothetically, a representation that is easily able to recover another representation may be preferable since it retains more information.

Results presented in Figure D.1 show the average across datasets of the cross-representation

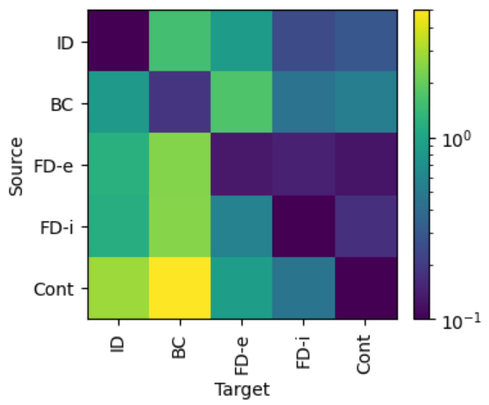


Figure D.1: Cross-representation prediction error of a small MLP on a validation set from the pretraining distribution. Results are normalized per dataset by the mean error on that dataset and then averaged across datasets.

prediction error on a validation set from the pretraining distribution (normalized by the mean prediction error on each dataset). There are several possible takeaways from this experiments. First, looking a the rows, which correspond to the error when each method is used as the source, we can see that inverse dynamics generally has the lowest average error for predicting the other representations. This suggests that inverse dynamics is doing a good job of recovering the information that is shared among all the representations. Second, looking at the columns, which correspond to error when each representation is used as the target, we see that BC is the most difficult to predict and inverse dynamics is second most difficult. This is a somewhat surprising result, but suggests that these representations actually contain information that may have been thrown away (or at made least difficult to access via small MLP) within the other representations. Finally, note that the contrastive learner is both the worst source and easiest target, which is consistent with the idea that those representations are losing important task-relevant information.

Full results on each dataset can be found in Appendix D.2.3 and full methodological details can be found in Appendix D.3.

D.2.2 Per dataset evaluation success results

In the main text and Section D.2.1 we have only presented aggregate results that average across datasets. These averages make it easier to summarize comparisons between methods, but they sacrifice the precision of how the results vary across datasets. In this section we present per dataset results for all of the relevant sweeps across dataset variations including pretraining size, finetuning size, and finetuning size when we ablate in distribution contexts or observability of the context in the observation.

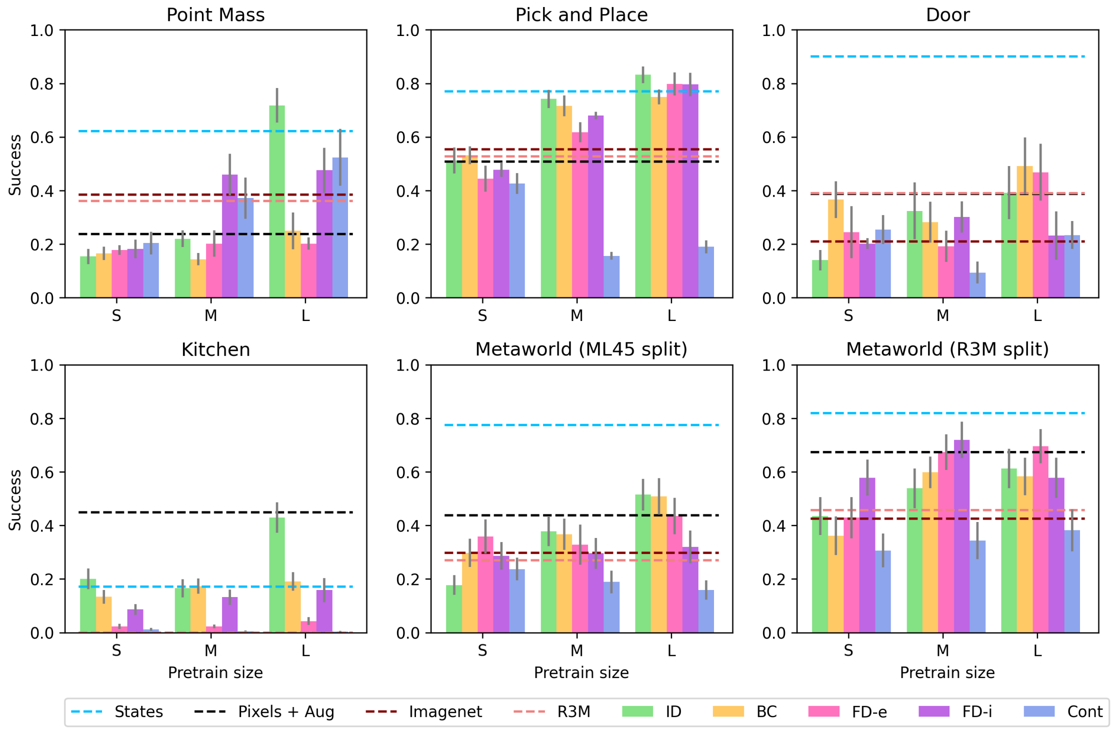


Figure D.2: The per dataset results of sweeping over pretraining size, corresponding to the right panel of Figure 6.3. Error bars show standard error over seeds and contexts (as described in Table 6.1). Horizontal lines indicate mean performance of algorithms that do not depend on pretraining size.

Pretraining size. First, we present the full ablation over pretraining size, corresponding to the right panel of Figure 6.3. The full per dataset results are shown in Figure D.2.

There are several findings in the dataset-specific results that are not visible in the aggregate reported in the main text:

- First, the kitchen environment is a clear outlier mainly due to the stochasticity in the data generating process and smaller dataset size compared to the others (see Appendix D.3.1 for more detailed description of the data). As a result of the noise added to the low dimensional states, training from States actually underperforms training from Pixels + Aug. We hypothesize that this is due to some implicit regularization that arises from training from the rendered noisy observations instead of the low dimensional noisy states. Importantly, inverse dynamics is much better able to handle the stochasticity than the alternative methods given the relatively small pretraining dataset and is the only method that is able to perform comparably to training from scratch.

- Point mass is the only environment where the externally pretrained representations (R3M and Imagenet) substantially outperform training from Pixels + Aug and they are substantially outperformed on kitchen and the metaworld datasets. We hypothesize that this shows how it is quite difficult to transfer features across domains and see consistent benefits on challenging tasks.

- Note that performance of contrastive learning is substantially better relative to the alternatives on point mass. We hypothesize that this is due to the fact that random crop augmentations are actually a reasonable simulation of the dynamics in the pointmass environment specifically so that contrastive learning becomes more similar to implicit forward dynamics.

Finetuning size. Next, we present the full ablation over finetuning size, corresponding to the left panel of Figure 6.3. The full per dataset results are shown in Figure D.3.

Again, as described above, Kitchen is a clear outlier due to stochasticity with inverse dynamics

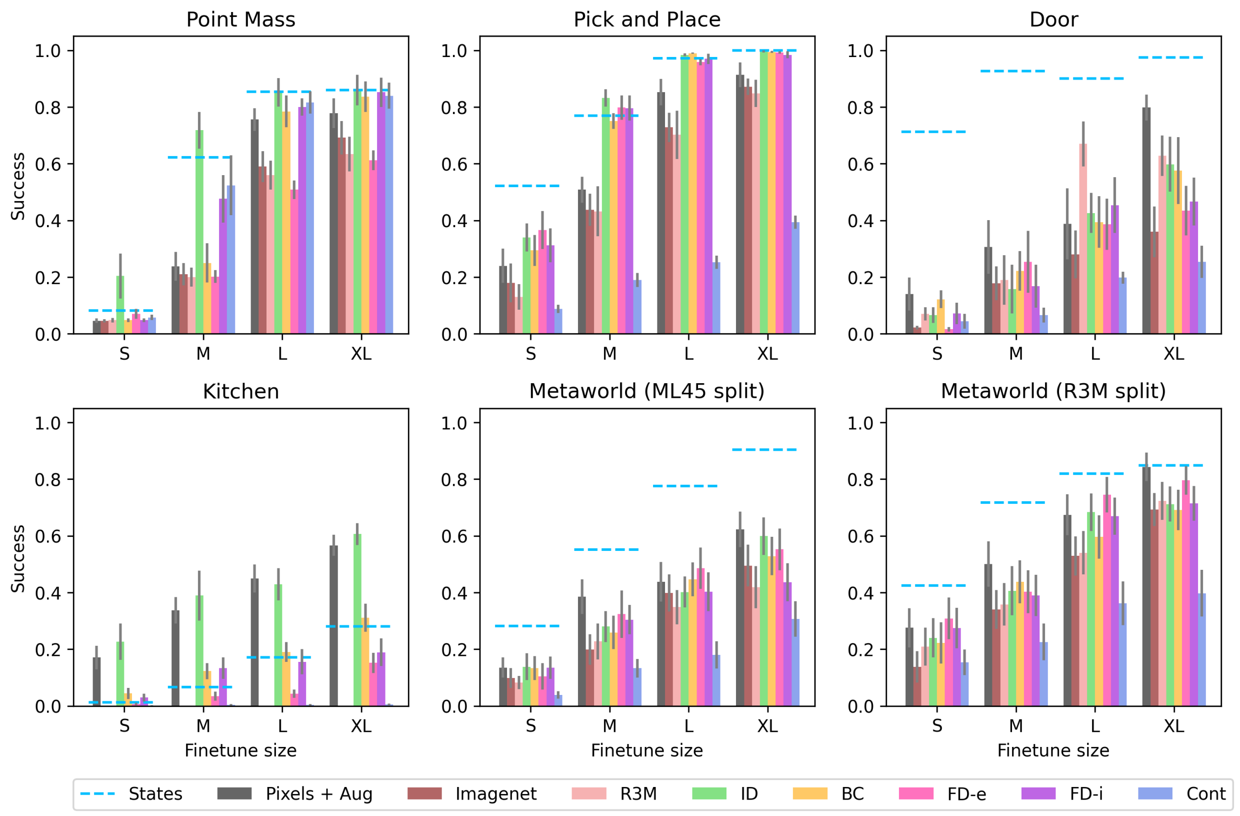


Figure D.3: The per dataset results of sweeping over finetuning size, corresponding to the left panel of Figure 6.3. Error bars show standard error over seeds and contexts (as described in Table 6.1).

the best performer. Inverse dynamics is also the clear winner on point mass and a slight winner on pick and place. The other tasks are more ambiguous with many methods performing about the same, and none substantially better than training from scratch (across all pretraining sizes). Disaggregating the results here shows how even though inverse dynamics is clearly the best in aggregate, this is not necessarily true on every dataset. As we will see in Figure D.4, we hypothesize that much of this weak performance can be attributed to the fact that the evaluation contexts in door and the two metaworld variants are truly out of distribution, making it difficult for any pretraining method to generalize.

Ablating in distribution contexts. Next, we present the full per dataset results when we ensure that all the evaluation contexts are included in the pretraining distribution, corresponding to Figure 6.4 in the main text. The full per dataset results are shown in Figure D.4.

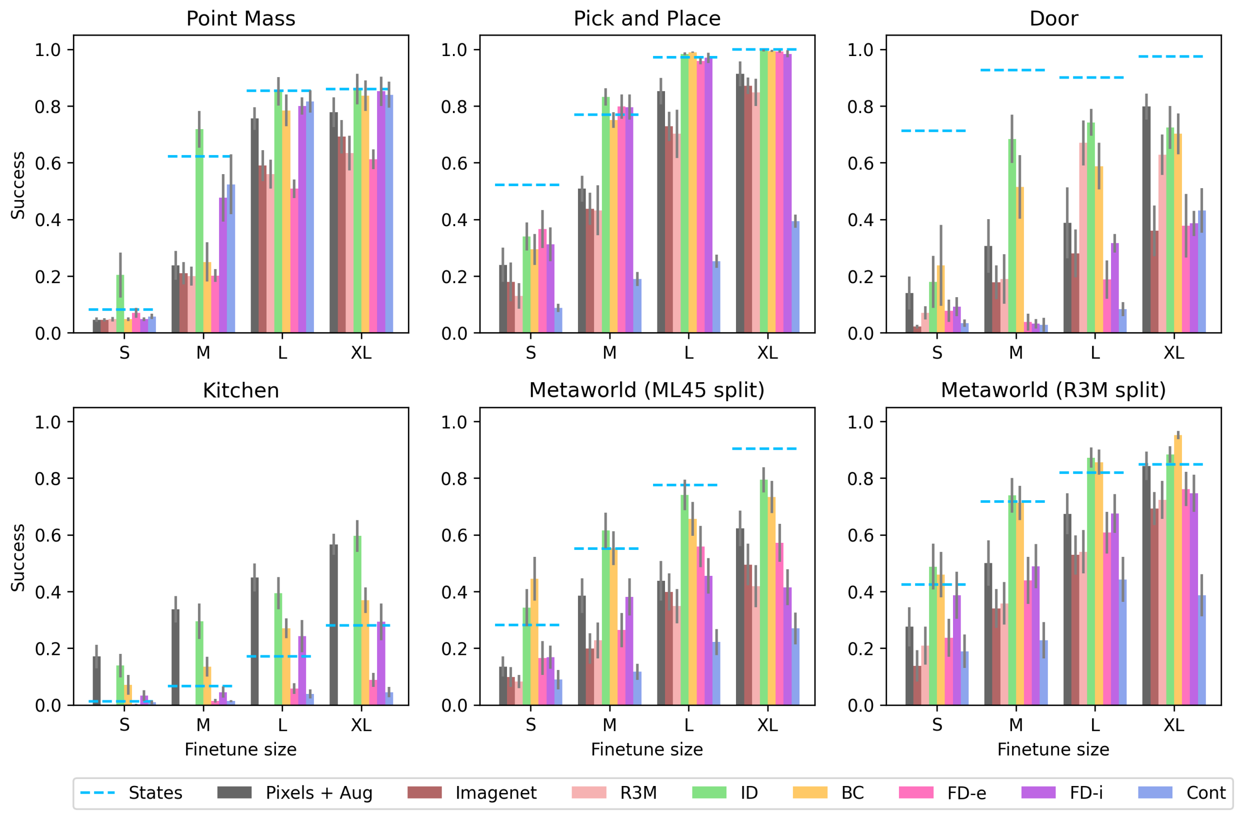


Figure D.4: The per dataset results of sweeping over finetuning size when we include the evaluation tasks in the pretraining data, corresponding to Figure 6.4. Error bars show standard error over seeds and contexts (as described in Table 6.1).

It is important to compare these results to those that include out of distribution evaluation contexts in Figure D.3. First, note that the evaluation contexts on point mass and pick and place were already in distribution, so they are kept the same. However, on door and the two metaworld splits there is a *substantial* improvement, especially for inverse dynamics and BC. This shows how these methods can benefit from being applied on tasks that are contained in the pretraining distribution. Interestingly, even though the evaluation contexts are now in distribution, the forward dynamics representations do not see substantial improvements and are still outperformed by training from scratch on the more challenging datasets.

Aggregating based on context observability. Finally, we present the full results for aggregations across whether the context is observable, corresponding to Figure 6.5 in the main text. Context is latent in point mass, pick and place, and kitchen, but inferrable in door and both metaworld splits. The results are shown in Figure D.5. Note that these results are just grouped averages over the results presented in Figure D.3.

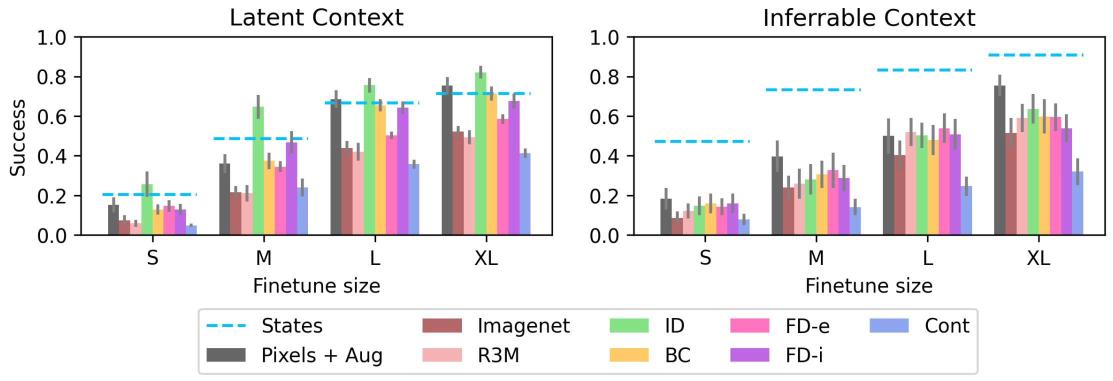


Figure D.5: The full results of aggregating based on the observability of the context variable, corresponding to Figure 6.5. Error bars show standard error over seeds and contexts (as described in Table 6.1) then averaged across datasets.

Compared to Figure 6.5, we now include the results from all algorithms and also from the environments where the context is inferrable. As reported in the main text, there is a clear gap between inverse dynamics and BC when the context is latent, likley due to confounding. Here we see that this gap largely disappears in the datasets where the context is inferrable and generally the disparities between methods shrink.

D.2.3 Per dataset representation analysis

Now we present the per dataset results of the various methods of representation analysis based on predicting different target quantities of interest: the action, the low dimensional state, and the other representations themselves.

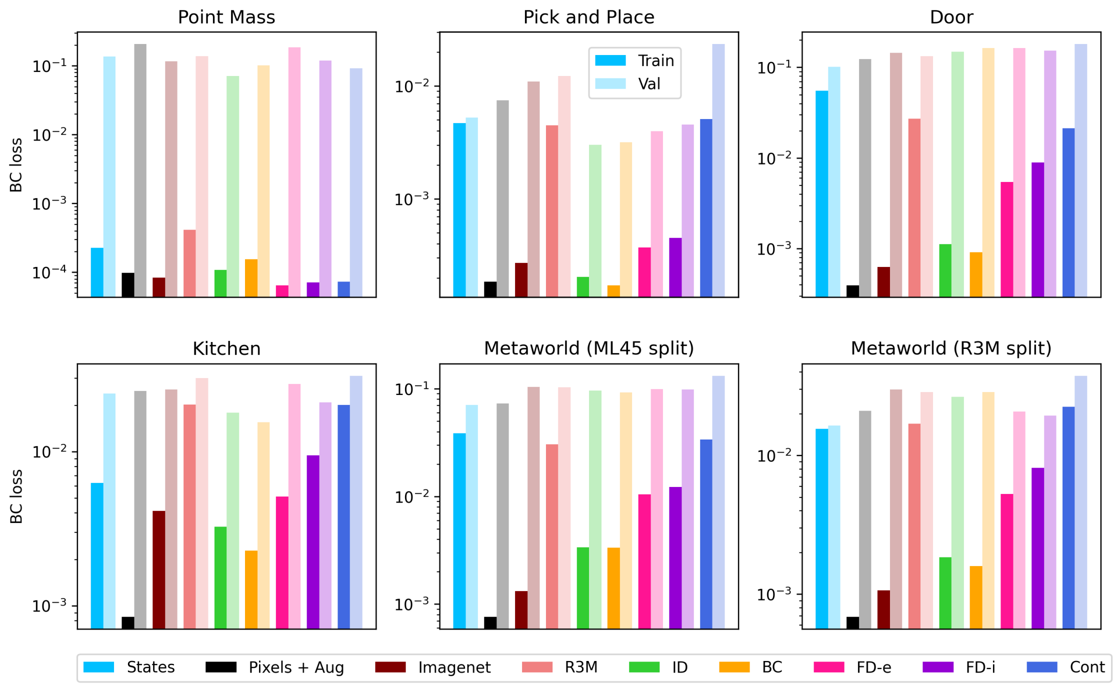


Figure D.6: Full per dataset results of action prediction on the finetuning distribution.

Predicting action. First we present the per dataset results for train and validation action prediction on the finetuning datasets using the default pretraining and finetuning size. These results correspond to Figure 6.6 from the main text. Unlike in the main text, here we do not do any normalization of the losses, so the losses occur at different scales on each dataset depending on how difficult the prediction task is. Results are shown in Figure D.6.

Predicting state. Next, we present the per dataset results for predicting the low dimensional state on the pretraining distribution from the various learned representations. These results correspond to Figure 6.7 in the main text. Again, unlike in the main text, results are not normalized, so they occur at different scales across environments. Results are shown in Figure D.7.

Note that as mentioned before, there is stochasticity added to the low dimensional states in the kitchen environment. This makes it difficult for any of the methods to substantially outperform the floor set by the noise level.

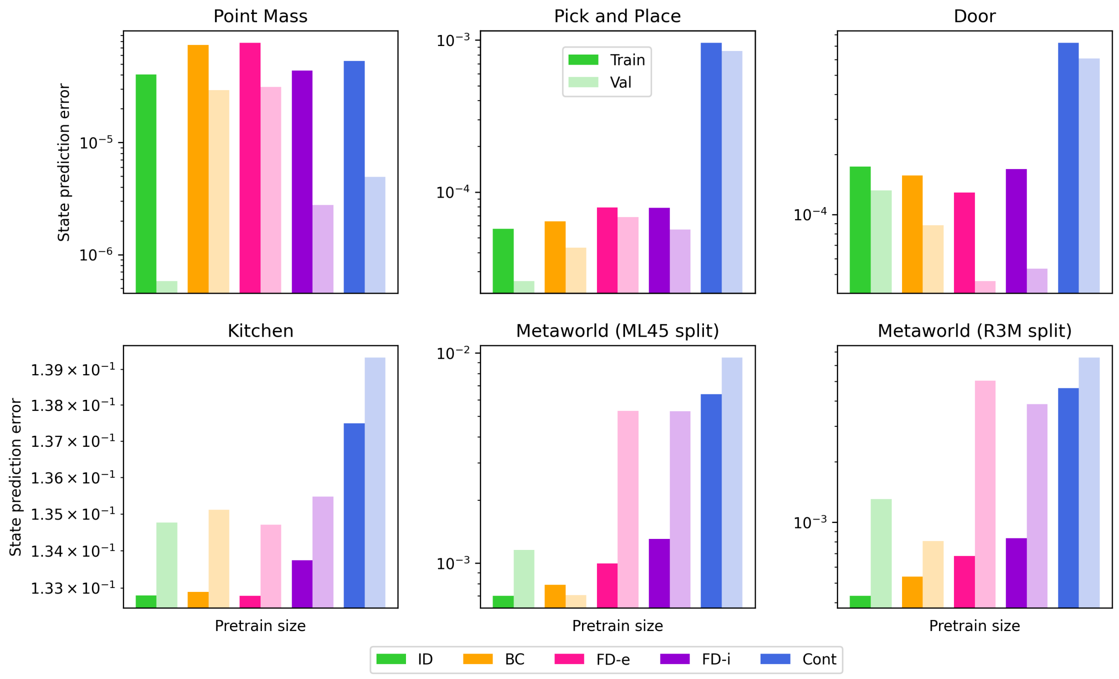


Figure D.7: Full per dataset results for state prediction on the pretraining distribution.

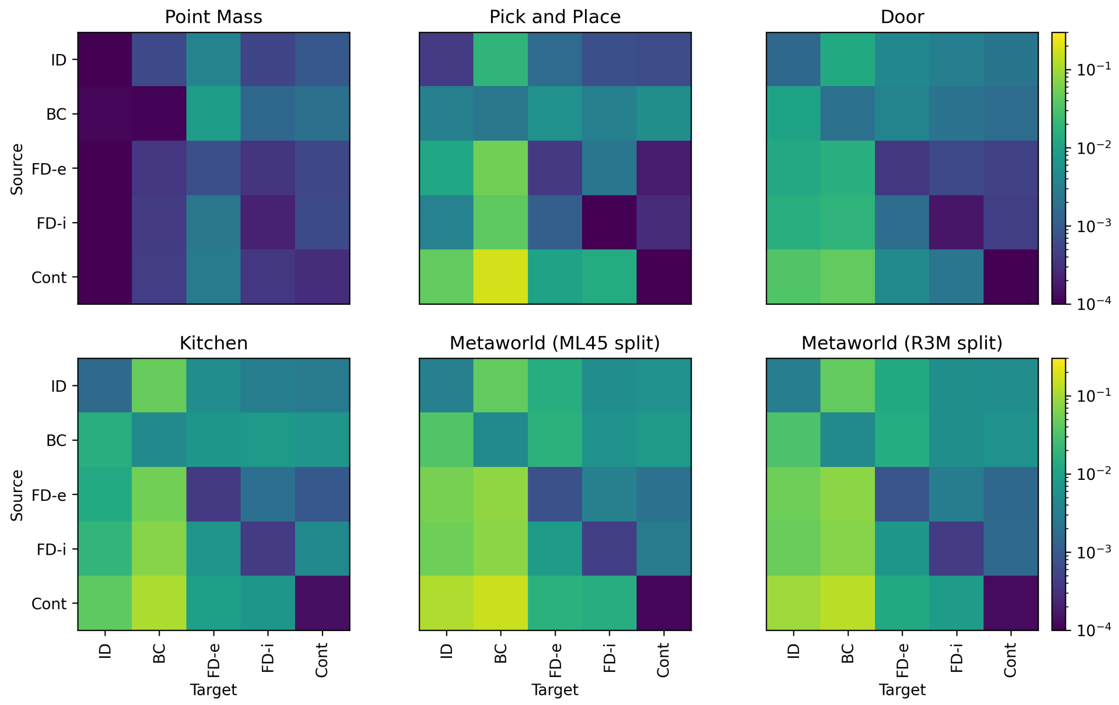


Figure D.8: Per dataset results for cross-representation prediction on the pretraining distribution. Color shows the validation error of predicting target from source.

Predicting across representations. Finally, we present the per dataset results for predicting across the different learned representations on the pretraining distribution. These results correspond to Figure D.1. Again, unlike in the averaged figure, this figure is not normalized, so the scales vary across datasets. We truncate the color scale at 1e-4 on the low end for easier visualization.

D.2.4 Ablation of multistep dynamics

As mentioned in the main text, some work argues for multistep dynamics models [Efroni et al. 2021; Lamb et al. 2022]. Note that this work focuses on settings with exogenous noise which are different from the simpler settings that we consider. To confirm that using multistep dynamics models does not help to learn better representations, we run an ablation of the number of steps included in the dynamics model on three environments: point mass, pick and place, and door and two algorithms: inverse dynamics and implicit forward dynamics. Results are shown in Figure D.9. At a high level, we basically find little difference when ablating the number of steps, so we default to using one step models everywhere for simplicity.

Note: for inverse dynamics models, we learn a k step model by predicting a_t given o_t and o_{t+k}. For forward dynamics, we learn a k step model by predicting o_{t+k} given o_t and $a_{t:t+k}$.

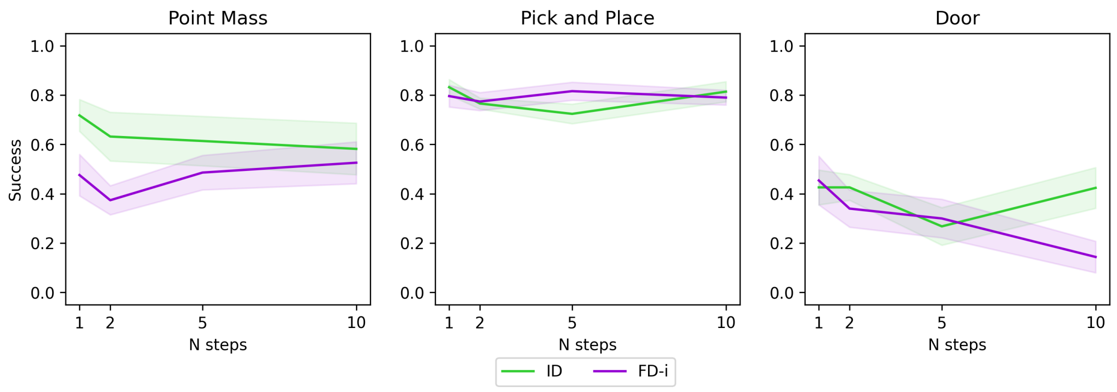


Figure D.9: Sweep over the number of timesteps included in the dynamics models.

D.3 Detailed experimental methodology

In this section we present a detailed account of out methodology. We also release our code that was used to perform the experiments for full transparency. We split up the description into Section D.3.1 which describes the environments and dataset generation, Section D.3.2 which describes the details of the pretraining pipeline, and Section D.3.3 which describes the details of the finetuning and evaluation pipeline.

All code is at `https://github.com/davidbrandfonbrener/imitation_pretraining`.

D.3.1 Envionment and dataset details

Software dependencies. All of our environments are based on the MuJoCo simulator [Todorov et al. 2012b]. The point mass environment is derived from the DM control suite [Tunyasuvunakool et al. 2020]. The kitchen environment and dataset was introduced in Gupta et al. [2019]. The rest of the environments are taken from Metaworld [Yu et al. 2020]. We describe each environment in detail and summarize the descriptions in Table D.1

Point mass. The point mass environment consists of an actuated point mass on a 2d plane. In our version, the context $c \in \mathbb{R}^2$ determines the goal location. Then, the demonstration policy π_c^* is a PD controller that moves the point from the current position x to the goal position c. Because the context variable is continuous, we sample an independent context for each trajectory in the pretraining dataset from the uniform distribution over possible goal states. The context is fully latent and not observable in the observation. The low dimensional state is the 2d position and the high dimensional images are 84x84x3.

Pick and place. The pick and place task is taken from the metaworld suite. In our version, the context $c \in \mathbb{R}^3$ determines the goal location for the block. The demonstration policy π_c^* is a scripted policy from the metaworld repo. We remove the goal indicator from the image in this environment so that the context is fully latent and not observable from the observation. The low dimensional state is the 3d position of the gripper, 1d openness of the gripper, and 7d position and orientation of the block. The high dimensional observations are images of size 120x120x3.

Door. The door environment is also taken from the metaworld suite. In our version, the context $c \in [4]$ determines the index of the environment from door-close, door-open, door-unlock, and door-lock. For our default experiments we use door-close, door-open, and door-unlock as the pretraining contexts and door-lock as the eval context. For the ablation where we ensure that the eval context is in the pretraining distribution, we include door-lock in the pretraining data. The demonstration policy π_c^* is a scripted policy from the metaworld repo. Given the context, the initial position of the robot, initial position of the door, and goal position (which is visible in the observation image) are all randomized. Note, the context is inferrable since the initial position of the door and lock allow the learner to infer the context. The low dimensional state is the 3d position of the gripper, 1d openness of the gripper, 7d position and orientation of two objects in the scene, and 3d goal position. The high dimensional observations are images of size 120x120x3.

Kitchen. The kitchen environment and dataset are taken from Gupta et al. [2019]. Each trajectory contains a sequence of four tasks in a simulated kitchen collected by a human demonstrator. In our version, the context $c \in [24]$ is determined by the sequence of four tasks contained within the demonstration trajectory (of which there are 24 possibilities). We evaluate on three contexts: microwave-kettle-light switch-slide cabinet, bottom burner-top burner-slide cabinet-hinge cabinet, and kettle-bottom burner-top burner-light switch. In our default setup, we pretrain on the other 21 contexts, and in the ablation of in distribution evaluation we pretrain on all 24 contexts. The context is fully latent and not observable from the initial state. The low dimensional state is a 9d description of the arm position and a 21d description of the position of objects in the kitchen. The high dimensional observations are images of size 120x120x3.

Note: the kitchen environment is the only one that we consider that has added noise. The raw data from Gupta et al. [2019] contains gaussian noise added to the low dimensional states and actions, so this noise cannot be removed without re-generating the data. We render the images from the noisy states, so there is also noise present in the image observations. We also evaluate in an environment with the same noise added, so there is no gap between training and eval.

Metaworld (ML45 and R3M). Finally, we consider two variants of the full metaworld suite. Here the context $c \in [50]$ determines which metaworld task is used. We consider two different train-eval splits for our default environments. The ML45 split takes the eval tasks from the original metaworld ML45 task which are bin-picking, box-close, hand-insert, door-lock, and door-unlock. The R3M split takes the eval tasks that were chosen in the R3M paper [Nair et al. 2022]: assembly, bin-picking, button-press, drawer-open, and hammer. Given the context, the initial and goal positions are randomized. The goal position is visible in the observation. The low dimensional state is the 3d position of the gripper, 1d openness of the gripper, 7d position and orientation of (potentially) two objects in the scene, and 3d goal position. The high dimensional observations are images of size 120x120x3.

Table D.1: A summary of the description of datasets above. Inferrable refers to whether the context is observable. OOD refers to whether the evaluation context is out of distribution.

Dataset	Policy	Context	Inferrable	OOD	Noise	State dim
Point mass	PD controller	$\mathbb{R}^2$	No	No	No	2
Pick and place	Script	$\mathbb{R}^3$	No	No	No	11
Door	Script	[4]	Yes	Yes	No	21
Kitchen	Human	[24]	No	Yes	Yes	30
Metaworld-ML45	Script	[50]	Yes	Yes	No	21
Metaworld-R3M	Script	[50]	Yes	Yes	No	21

D.3.2 Pretraining details

Software dependencies. We implement all of our training in JAX [Bradbury et al. 2018]. We use flax for neural networks [Heek et al. 2020] and optax for optimization [Babuschkin et al. 2020]. Our code is loosely based on Kostrikov [2022].

Architecture. All of our pretraining algorithms share exactly the same encoder architecture to ensure that we have a fair comparison. Since our tasks are relatively simple visually, and so as to allow for large scale experiments without too much compute, we use a relatively small convnet encoder. Specifically, we follow the architecture from Yarats et al. [2021b] which consists of a 4 layer convnet with 3x3 filters, number of channels of (32, 64, 128, 256), and strides of (2,2,1,1). We add a modification to include a spatial softmax activation [Finn et al. 2016], which we found to be important for the manipulation tasks we consider. This is followed by a linear layer to project into the embedding dimension of 64 and finally a layernorm and tanh activation to normalize the embedding. We use the gelu activation function throughout.

Now we will birefly describe the architecture used for each pretraining algorithm, following their descriptions in Section 6.4.2:

- Inverse dynamics: the inverse dynamics head is an MLP that takes in $\phi(o), \phi(o')$ and produces an estimated action. This MLP has two hidden layers of width 256 and dropout of 0.1 during training.
- BC: the BC policy head is an MLP with two hidden layers of width 256 and dropout of 0.1 during training.
- Implicit forward dynamics: the implicit forward dynamics model uses an action encoder $\phi_a(a)$ which outputs a 64 dimensional normalized action embedding which is concatenated to $\phi(o)$ to form $\phi(o, a)$. Then there are two projection heads f_1, f_2 that take in $\phi(o, a)$ and

$\phi(o')$ respectively and produce 64 dimensional embeddings that are normalized to have unit norm. All these networks (ϕ_a, f_1, and f_2) are MLPs with two hidden layers of width 256 and the relevant input and output dimensions.

- Explicit forward dynamics: the explicit forward dynamics model uses the same architecture to encode a with ϕ_a. Then, instead of projection heads, we require a convolutional decoder to produce an image. Following Yarats et al. [2021b] we use an architecture that inverts the encoder, having a dense projection layer followed by channels of (256, 128, 64, 32) and strides of (1,1,2,2).

- Contrastive: the contrastive network is the same as the implicit forward dynamics network except that there is no action input and o' is replaced by an augmentation of o.

Training hyperparameters. For pretraining, we split the datasets into two categories: easy (point mass, pick and place, and door) and hard (kitchen, metaworld-ml45, and meatworld-r3m). On the easy tasks we train for 100k gradient steps and on the hard tasks we train for 200k gradient steps. Batch size is 256 for all methods except explicit forward dynamics where (due to the added compute required for the decoder) we use batch size of 128 to even out computational requirements across methods. All methods are trained with the adamw optimizer with learning rate 1e-3, a cosine learning rate decay schedule, and default weight decay of 1e-4.

Data augmentation. Following [Chen et al. 2022] and others, we note that cropping augmentations are the most important for training policies in simulated visual domains. As such, all of our pretraining algorithms (and the Pixels + Aug baseline) use random cropping augmentations, and we found this to be an important implementation detail. The one exception is explicit forward dynamics where we found it difficult to reconstruct images with augmentations, so we omit them for that algorithm.

Compute resources. Pretraining was all done on an internal cluster using RTX8000 GPUs. We estimate that the final training run needed to produce the results in the paper took approximately 600 GPU hours.

D.3.3 Finetuning and evaluation details

Training hyperparameters. The policy is always an MLP with two hidden layers of width 256. We use gelu activation and apply dropout with probability 0.1 during finetuning. We finetune on every dataset for 10k gradient steps with batch size 256. All policies are trained with the adamw optimizer with learning rate 1e-3, a cosine learning rate decay schedule, and default weight decay of 1e-4.

As explained in Table 6.1 there are several seeds and evaluation contexts for each environment. For example, for the default results in Figure 6.1 we end up having a total of 80 different finetuning datasets per representation when sweeping across dataset, context, and seed so that Figure 6.1 is reporting aggregate results across 720 finetuning and evaluation runs.

Evaluation hyperparameters. Each evaluation is run for 100 episodes in the environment to estimate the success of the policy (except for the kitchen environment where we run 50 episodes due to slow rendering of that environment).

Compute resources. Finetuning and evaluation was all done on an internal cluster on CPU (since the finetuned policy network is small and environments run on CPU). We estimate that all the finetuning and evaluation in the final runs used to produce results for the paper took approximately 2000 CPU hours.

D.4 Extended analysis discussion

Here we provide a more detailed discussion of related theoretical work.

One recent line of work focuses on learning representations for exploration [Efroni et al. 2021; Lamb et al. 2022] and offline RL [Islam et al. 2022] in the presence of *exogenous noise*. The exogenous noise setting means that the high dimensional observations contain information that is not effected by the actions; e.g., background dynamics that appear in image observations but do not affect the task. This line of work argues that inverse dynamics modeling is the best approach to ignore exogenous noise. Our results are complementary to this line of work in showing that even in settings *without* exogenous noise, inverse dynamics is still often preferable to alternatives for representation learning. Moreover, we consider a *multitask* imitation setting with latent contexts while they consider single task and reward-directed problems.

Another line of work proves that learning a forward dynamics model is a well-motivated approach for multitask imitation [Nachum and Yang 2021]. While that work does not directly compare to inverse dynamics pretraining, we find that inverse dynamics pretraining outperforms forward dynamics modeling in our settings. Moreover, while this paper shows that if our representation learns a good forward dynamics model that it works well for imitation, it does not discuss how efficiently such a representation can be learned. So, while both methods are well-motivated, we find inverse dynamics modeling to be more efficient than learning the forward dynamics.

Finally, another line of work studies multitask representation learning for imitation by directly performing behavior cloning [Arora et al. 2019; Zhang et al. 2022]. These methods provide positive results for the approach, but require algorithms that have access to the latent context information which must be discrete so as to learn a separate policy for every pretraining context, thus avoiding confounding. This method requires extra information and is difficult to scale to very large numbers of contexts. In contrast, we find that inverse dynamics modeling is able to perform well without this extra information or added complexity of learning multiple models and naturally avoids confounding by the latent context information.

Bibliography

Achiam, J., Held, D., Tamar, A., and Abbeel, P. (2017). Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31. PMLR.

Achiam, J., Knight, E., and Abbeel, P. (2019). Towards characterizing divergence in deep q-learning. *arXiv preprint arXiv:1903.08894*.

Agarwal, A., Jiang, N., and Kakade, S. (2019). Reinforcement learning: Theory and algorithms.

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. (2022). Reincarnating reinforcement learning: Reusing prior computation to accelerate progress. *Advances in Neural Information Processing Systems*, 35:28955–28971.

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., et al. (2022). Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

Ajay, A., Kumar, A., Agrawal, P., Levine, S., and Nachum, O. (2020). Opal: Offline primitive discovery for accelerating offline reinforcement learning. *arXiv preprint arXiv:2010.13611*.

Amortila, P., Jiang, N., and Xie, T. (2020). A variant of the wang-foster-kakade lower bound for the discounted setting. *ArXiv*, abs/2011.01075.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., and Zaremba, W. (2017). Hindsight experience replay. *Advances in neural information processing systems*, 30.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.

Arora, S., Du, S., Kakade, S., Luo, Y., and Saunshi, N. (2020). Provable representation learning for imitation learning via bi-level optimization. In *International Conference on Machine Learning*, pages 367–376. PMLR.

Arora, S., Khandeparkar, H., Khodak, M., Plevrakis, O., and Saunshi, N. (2019). A theoretical analysis of contrastive unsupervised representation learning. *arXiv preprint arXiv:1902.09229*.

Aytar, Y., Pfaff, T., Budden, D., Paine, T., Wang, Z., and De Freitas, N. (2018). Playing hard exploration games by watching youtube. *Advances in neural information processing systems*, 31.

Babuschkin, I., Baumli, K., Bell, A., Bhupatiraju, S., Bruce, J., Buchlovsky, P., Budden, D., Cai, T., Clark, A., Danihelka, I., Dedieu, A., Fantacci, C., Godwin, J., Jones, C., Hemsley, R., Hennigan, T., Hessel, M., Hou, S., Kapturowski, S., Keck, T., Kemaev, I., King, M., Kunesch, M., Martens, L., Merzic, H., Mikulik, V., Norman, T., Papamakarios, G., Quan, J., Ring, R., Ruiz, F., Sanchez, A., Schneider, R., Sezener, E., Spencer, S., Srinivasan, S., Stokowiec, W., Wang, L., Zhou, G., and Viola, F. (2020). The DeepMind JAX Ecosystem.

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.

Bartlett, P. L., Jordan, M. I., and McAuliffe, J. D. (2006). Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156.

Bellemare, M. G., Dabney, W., and Rowland, M. (2022). *Distributional Reinforcement Learning*. MIT Press. http://www.distributional-rl.org.

Boucheron, S., Bousquet, O., and Lugosi, G. (2005). Theory of classification: A survey of some recent advances. *ESAIM: probability and statistics*, 9:323–375.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.

Brandfonbrener, D., Bietti, A., Buckman, J., Laroche, R., and Bruna, J. (2022a). When does return-conditioned supervised learning work for offline reinforcement learning? *arXiv preprint arXiv:2206.01079*.

Brandfonbrener, D. and Bruna, J. (2020). Geometric insights into the convergence of nonlinear td learning. In *International Conference on Learning Representations*.

Brandfonbrener, D., Nachum, O., and Bruna, J. (2023). Inverse dynamics pretraining learns good representations for multitask imitation. *arXiv preprint arXiv:2305.16985*.

Brandfonbrener, D., Tu, S., Singh, A., Welker, S., Boodoo, C., Matni, N., and Varley, J. (2022b). Visual backtracking teleoperation: A data collection protocol for offline image-based reinforcement learning. *arXiv preprint arXiv:2210.02343*.

Brandfonbrener, D., Whitney, W. F., Ranganath, R., and Bruna, J. (2021). Offline rl without off-policy evaluation. In *Advances in Neural Information Processing Systems*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *CoRR*, abs/1606.01540.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Dabis, J., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Hsu, J., et al. (2022). Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Buckman, J., Gelada, C., and Bellemare, M. G. (2020). The importance of pessimism in fixed-dataset policy optimization.

Burda, Y., Edwards, H., Storkey, A. J., and Klimov, O. (2019). Exploration by random network distillation. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Burkhardt, J. (2014). The truncated normal distribution.

Cabi, S., Colmenarejo, S. G., Novikov, A., Konyushkova, K., Reed, S., Jeong, R., Zolna, K., Aytar, Y., Budden, D., Vecerik, M., et al. (2019). Scaling data-driven robotics with reward sketching and batch reinforcement learning. *arXiv preprint arXiv:1909.12200*.

Chen, A. S., Nam, H., Nair, S., and Finn, C. (2021a). Batch exploration with examples for scalable robotic reinforcement learning. *IEEE Robotics and Automation Letters*, 6(3):4401–4408.

Chen, J. and Jiang, N. (2019). Information-theoretic considerations in batch reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. (2021b). Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. (2021c). Decision transformer: Reinforcement learning via sequence modeling. *CoRR*, abs/2106.01345.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020a). A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR.

Chen, X., Toyer, S., Wild, C., Emmons, S., Fischer, I., Lee, K.-H., Alex, N., Wang, S. H., Luo, P., Russell, S., et al. (2022). An empirical investigation of representation learning for imitation. *arXiv preprint arXiv:2205.07886*.

Chen, X., Zhou, Z., Wang, Z., Wang, C., Wu, Y., and Ross, K. (2020b). Bail: Best-action imitation learning for batch deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33.

Consortium, T. U. (2020). UniProt: the universal protein knowledgebase in 2021. *Nucleic Acids Research*, 49(D1):D480–D489.

Cui, Z. J., Wang, Y., Shafiullah, N. M. M., and Pinto, L. (2022). From play to policy: Conditional behavior generation from uncurated robot data. *arXiv e-prints*, pages arXiv–2210.

Dean, S. and Recht, B. (2021). Certainty equivalent perception-based control. In *Learning for Dynamics and Control*, pages 399–411. PMLR.

Dendorfer, P., Osep, A., and Leal-Taixé, L. (2020). Goal-gan: Multimodal trajectory prediction based on goal position estimation. In *Proceedings of the Asian Conference on Computer Vision*.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Li, F. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ding, Y., Florensa, C., Abbeel, P., and Phielipp, M. (2019). Goal-conditioned imitation learning. *Advances in neural information processing systems*, 32.

Duan, Y., Andrychowicz, M., Stadie, B., Jonathan Ho, O., Schneider, J., Sutskever, I., Abbeel, P., and Zaremba, W. (2017). One-shot imitation learning. *Advances in neural information processing systems*, 30.

Duan, Y., Jia, Z., and Wang, M. (2020). Minimax-optimal off-policy evaluation with linear function approximation. In *International Conference on Machine Learning*, pages 2701–2709. PMLR.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2019). Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.

Efroni, Y., Misra, D., Krishnamurthy, A., Agarwal, A., and Langford, J. (2021). Provable rl with exogenous distractors via multistep inverse dynamics. *arXiv preprint arXiv:2110.08847*.

Emmons, S., Eysenbach, B., Kostrikov, I., and Levine, S. (2021). Rvs: What is essential for offline rl via supervised learning? *arXiv preprint arXiv:2112.10751*.

Endrawis, S., Leibovich, G., Jacob, G., Novik, G., and Tamar, A. (2021). Efficient self-supervised data collection for offline robot learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4650–4656. IEEE.

Eysenbach, B., Geng, X., Levine, S., and Salakhutdinov, R. R. (2020). Rewriting history with inverse rl: Hindsight inference for policy improvement. *Advances in neural information processing systems*, 33:14783–14795.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. (2019). Diversity is all you need: Learning skills without a reward function. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Eysenbach, B., Zhang, T., Salakhutdinov, R., and Levine, S. (2022). Contrastive learning as goal-conditioned reinforcement learning. *arXiv preprint arXiv:2206.07568*.

Finn, C., Abbeel, P., and Levine, S. (2017a). Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

Finn, C., Tan, X. Y., Duan, Y., Darrell, T., Levine, S., and Abbeel, P. (2016). Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE.

Finn, C., Yu, T., Zhang, T., Abbeel, P., and Levine, S. (2017b). One-shot visual imitation learning via meta-learning. In *Conference on robot learning*, pages 357–368. PMLR.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. (2020). D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*.

Fu, J., Norouzi, M., Nachum, O., Tucker, G., Wang, Z., Novikov, A., Yang, M., Zhang, M. R., Chen, Y., Kumar, A., et al. (2021a). Benchmarks for deep off-policy evaluation. *arXiv preprint arXiv:2103.16596*.

Fu, X., Yang, G., Agrawal, P., and Jaakkola, T. (2021b). Learning task informed abstractions. In *International Conference on Machine Learning*, pages 3480–3491. PMLR.

Fujimoto, S., Conti, E., Ghavamzadeh, M., and Pineau, J. (2019a). Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint arXiv:1910.01708*.

Fujimoto, S. and Gu, S. S. (2021). A minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2106.06860*.

Fujimoto, S., Meger, D., and Precup, D. (2018a). Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*.

Fujimoto, S., Meger, D., and Precup, D. (2019b). Off-policy deep reinforcement learning without exploration. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2052–2062. PMLR.

Fujimoto, S., Meger, D., and Precup, D. (2019c). Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pages 2052–2062. PMLR.

Fujimoto, S., van Hoof, H., and Meger, D. (2018b). Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*.

Furuta, H., Matsuo, Y., and Gu, S. S. (2021). Generalized decision transformer for offline hindsight information matching. *arXiv preprint arXiv:2111.10364*.

Gelada, C., Kumar, S., Buckman, J., Nachum, O., and Bellemare, M. G. (2019). Deepmdp: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*, pages 2170–2179. PMLR.

Ghosh, D., Bhateja, C., and Levine, S. (2023). Reinforcement learning from passive data via latent intentions. *arXiv preprint arXiv:2304.04782*.

Ghosh, D., Gupta, A., and Levine, S. (2018). Learning actionable representations with goal-conditioned policies. *arXiv preprint arXiv:1811.07819*.

Ghosh, D., Gupta, A., Reddy, A., Fu, J., Devin, C., Eysenbach, B., and Levine, S. (2019). Learning to reach goals via iterated supervised learning. *arXiv preprint arXiv:1912.06088*.

Goo, W. and Niekum, S. (2020). You only evaluate once – a simple baseline algorithm for offline rl. In *Offline Reinforcement Learning Workshop at Neural Information Processing Systems*.

Grauman, K., Westbury, A., Byrne, E., Chavis, Z., Furnari, A., Girdhar, R., Hamburger, J., Jiang, H., Liu, M., Liu, X., et al. (2022). Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18995–19012.

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.

Gulcehre, C., Colmenarejo, S. G., Wang, Z., Sygnowski, J., Paine, T., Zolna, K., Chen, Y., Hoffman, M., Pascanu, R., and de Freitas, N. (2021). Regularized behavior value estimation. *arXiv preprint arXiv:2103.09575*.

Gulcehre, C., Wang, Z., Novikov, A., Paine, T. L., Colmenarejo, S. G., Zolna, K., Agarwal, R., Merel, J., Mankowitz, D., Paduraru, C., et al. (2020). Rl unplugged: Benchmarks for offline reinforcement learning. *arXiv preprint arXiv:2006.13888*.

Gupta, A., Kumar, V., Lynch, C., Levine, S., and Hausman, K. (2019). Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*.

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 *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. (2022). Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Heek, J., Levskaya, A., Oliver, A., Ritter, M., Rondepierre, B., Steiner, A., and van Zee, M. (2020). Flax: A neural network library and ecosystem for JAX.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. *Advances in neural information processing systems*, 29.

Hoque, R., Balakrishna, A., Putterman, C., Luo, M., Brown, D. S., Seita, D., Thananjeyan, B., Novoseller, E., and Goldberg, K. (2021). Lazydagger: Reducing context switching in interactive imitation learning. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 502–509. IEEE.

Hoque, R., Chen, L. Y., Sharma, S., Dharmarajan, K., Thananjeyan, B., Abbeel, P., and Goldberg, K. (2022). Fleet-dagger: Interactive robot fleet learning with scalable human supervision. *arXiv preprint arXiv:2206.14349*.

Islam, R., Tomar, M., Lamb, A., Efroni, Y., Zang, H., Didolkar, A., Misra, D., Li, X., van Seijen, H., Combes, R. T. d., et al. (2022). Agent-controller representations: Principled offline rl with rich exogenous information. *arXiv preprint arXiv:2211.00164*.

Jang, E., Irpan, A., Khansari, M., Kappler, D., Ebert, F., Lynch, C., Levine, S., and Finn, C. (2022). Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR.

Janner, M., Li, Q., and Levine, S. (2021). Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34.

Jaques, N., Ghandeharioun, A., Shen, J. H., Ferguson, C., Lapedriza, A., Jones, N., Gu, S., and Picard, R. (2019). Way off-policy batch deep reinforcement learning of implicit human preferences in dialog.

Jin, C., Krishnamurthy, A., Simchowitz, M., and Yu, T. (2020). Reward-free exploration for reinforcement learning. In *International Conference on Machine Learning*, pages 4870–4879. PMLR.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589.

Kaelbling, L. P. (1993). Learning to achieve goals. In *IJCAI*.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274.

Kalashnikov, D., Varley, J., Chebotar, Y., Swanson, B., Jonschkowski, R., Finn, C., Levine, S., and Hausman, K. (2021). Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Kelly, M., Sidrane, C., Driggs-Campbell, K., and Kochenderfer, M. J. (2019). Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8077–8083. IEEE.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kostrikov, I. (2021). Jaxrl: Implementations of reinforcement learning algorithms in jax., 10 2021. *URL https://github. com/ikostrikov/jaxrl*.

Kostrikov, I. (2022). JAXRL: Implementations of Reinforcement Learning algorithms in JAX. v2.

Kostrikov, I., Nachum, O., and Tompson, J. (2019). Imitation learning via off-policy distribution matching. *arXiv preprint arXiv:1912.05032*.

Kostrikov, I., Nair, A., and Levine, S. (2021a). Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*.

Kostrikov, I., Tompson, J., Fergus, R., and Nachum, O. (2021b). Offline reinforcement learning with fisher divergence critic regularization. *arXiv preprint arXiv:2103.08050*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114.

Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. (2019a). Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11761–11771.

Kumar, A., Hong, J., Singh, A., and Levine, S. (2021a). Should i run offline reinforcement learning or behavioral cloning? In *Deep RL Workshop NeurIPS 2021*.

Kumar, A., Peng, X. B., and Levine, S. (2019b). Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*.

Kumar, A., Singh, A., Tian, S., Finn, C., and Levine, S. (2021b). A workflow for offline model-free robotic reinforcement learning. *arXiv preprint arXiv:2109.10813*.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. (2020). Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*.

Lamb, A., Islam, R., Efroni, Y., Didolkar, A., Misra, D., Foster, D., Molu, L., Chari, R., Krishnamurthy, A., and Langford, J. (2022). Guaranteed discovery of controllable latent states with multi-step inverse models. *arXiv preprint arXiv:2207.08229*.

Lambert, N., Wulfmeier, M., Whitney, W., Byravan, A., Bloesch, M., Dasagi, V., Hertweck, T., and Riedmiller, M. (2022). The challenges of exploration for offline reinforcement learning. *arXiv preprint arXiv:2201.11861*.

Laroche, R., Trichelair, P., and Des Combes, R. T. (2019). Safe policy improvement with baseline bootstrapping. In *International Conference on Machine Learning*, pages 3652–3661. PMLR.

Laskin, M., Srinivas, A., and Abbeel, P. (2020). Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pages 5639–5650. PMLR.

Laskin, M., Yarats, D., Liu, H., Lee, K., Zhan, A., Lu, K., Cang, C., Pinto, L., and Abbeel, P. (2021). Urlb: Unsupervised reinforcement learning benchmark. *arXiv preprint arXiv:2110.15191*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Lee, J. N., Tucker, G., Nachum, O., and Dai, B. (2021). Model selection in batch policy optimization. *arXiv preprint arXiv:2112.12320*.

Lee, K.-H., Nachum, O., Zhang, T., Guadarrama, S., Tan, J., and Yu, W. (2022). Pi-ars: Accelerating evolution-learned visual-locomotion with predictive information representations. In *2022*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 1447–1454. IEEE.

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*.

Levine, S. (2021). Understanding the world through action. *arXiv preprint arXiv:2110.12543*.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Li, A. C., Pinto, L., and Abbeel, P. (2020). Generalized hindsight for reinforcement learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Liu, H. and Abbeel, P. (2021a). Aps: Active pretraining with successor features. In *International Conference on Machine Learning*, pages 6736–6747. PMLR.

Liu, H. and Abbeel, P. (2021b). Behavior from the void: Unsupervised active pre-training. *arXiv preprint arXiv:2103.04551*.

Lynch, C., Khansari, M., Xiao, T., Kumar, V., Tompson, J., Levine, S., and Sermanet, P. (2020). Learning latent plans from play. In *Conference on robot learning*, pages 1113–1132. PMLR.

Ma, Y. J., Sodhani, S., Jayaraman, D., Bastani, O., Kumar, V., and Zhang, A. (2022). Vip: Towards universal visual reward and representation via value-implicit pre-training. *arXiv preprint arXiv:2210.00030*.

Mandlekar, A., Xu, D., Mart'in-Mart'in, R., Zhu, Y., Fei-Fei, L., and Savarese, S. (2020). Human-in-the-loop imitation learning using remote teleoperation. *ArXiv*, abs/2012.06733.

Mathieu, M., Ozair, S., Srinivasan, S., Gulcehre, C., Zhang, S., Jiang, R., Le Paine, T., Zolna, K., Powell, R., Schrittwieser, J., et al. (2021). Starcraft ii unplugged: Large scale offline reinforcement learning. In *Deep RL Workshop NeurIPS 2021*.

Mendonca, R., Rybkin, O., Daniilidis, K., Hafner, D., and Pathak, D. (2021). Discovering and achieving goals via world models. *Advances in Neural Information Processing Systems*, 34.

Mhammedi, Z., Foster, D. J., Simchowitz, M., Misra, D., Sun, W., Krishnamurthy, A., Rakhlin, A., and Langford, J. (2020). Learning the linear quadratic regulator from nonlinear observations. *Advances in Neural Information Processing Systems*, 33:14532–14543.

Mitchell, E., Rafailov, R., Peng, X. B., Levine, S., and Finn, C. (2021). Offline meta-reinforcement learning with advantage weighting. In *International Conference on Machine Learning*, pages 7780–7791. PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. (2016). Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062.

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

Nachum, O. and Yang, M. (2021). Provable representation learning for imitation with contrastive fourier features. *Advances in Neural Information Processing Systems*, 34:30100–30112.

Nadjahi, K., Laroche, R., and des Combes, R. T. (2019). Safe policy improvement with soft baseline bootstrapping.

Nair, A., Gupta, A., Dalal, M., and Levine, S. (2020). Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*.

Nair, S., Rajeswaran, A., Kumar, V., Finn, C., and Gupta, A. (2022). R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*.

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

Paine, T. L., Paduraru, C., Michi, A., Gulcehre, C., Zolna, K., Novikov, A., Wang, Z., and de Freitas, N. (2020). Hyperparameter selection for offline reinforcement learning.

Pari, J., Shafiullah, N. M., Arunachalam, S. P., and Pinto, L. (2021). The surprising effectiveness of representation learning for visual imitation. *arXiv preprint arXiv:2112.01511*.

Paster, K., McIlraith, S., and Ba, J. (2022). You can't count on luck: Why decision transformers fail in stochastic environments. *arXiv preprint arXiv:2205.15967*.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR.

Pathak, D., Gandhi, D., and Gupta, A. (2019). Self-supervised exploration via disagreement. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 5062–5071. PMLR.

Peng, X. B., Kumar, A., Zhang, G., and Levine, S. (2019). Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*.

Pomerleau, D. A. (1988). Alvinn: An autonomous land vehicle in a neural network. In *NIPS*.

Pomerleau, D. A. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97.

Puterman, M. L. and Shin, M. C. (1978). Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137.

Quinonero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. (2008). *Dataset shift in machine learning*. Mit Press.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. (2021a). Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021b). Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, volume 139, pages 8748–8763.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. (2017). Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087.*

Rakelly, K., Zhou, A., Finn, C., Levine, S., and Quillen, D. (2019). Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR.

Riedmiller, M., Springenberg, J. T., Hafner, R., and Heess, N. (2021). Collect & infer–a fresh look at data-efficient reinforcement learning. *arXiv preprint arXiv:2108.10273.*

Riedmiller, M. A., Hafner, R., Lampe, T., Neunert, M., Degrave, J., de Wiele, T. V., Mnih, V., Heess, N., and Springenberg, J. T. (2018). Learning by playing solving sparse reward tasks from scratch. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4341–4350. PMLR.

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.

Schaal, S. (1996). Learning from demonstration. *Advances in neural information processing systems*, 9.

Scherrer, B., Gabillon, V., Ghavamzadeh, M., and Geist, M. (2012). Approximate modified policy iteration. *arXiv preprint arXiv:1205.3054.*

Schmidhuber, J. (2019). Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875.*

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897.

Schwarzer, M., Rajkumar, N., Noukhovitch, M., Anand, A., Charlin, L., Hjelm, R. D., Bachman, P., and Courville, A. C. (2021). Pretraining representations for data-efficient reinforcement learning. *Advances in Neural Information Processing Systems*, 34:12686–12699.

Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., and Pathak, D. (2020). Planning to explore via self-supervised world models. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 8583–8592. PMLR.

Seo, Y., Hafner, D., Liu, H., Liu, F., James, S., Lee, K., and Abbeel, P. (2023). Masked world models for visual control. In *Conference on Robot Learning*, pages 1332–1344. PMLR.

Seo, Y., Lee, K., James, S. L., and Abbeel, P. (2022). Reinforcement learning with action-free pre-training from videos. In *International Conference on Machine Learning*, pages 19561–19579. PMLR.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Siegel, N., Springenberg, J. T., Berkenkamp, F., Abdolmaleki, A., Neunert, M., Lampe, T., Hafner, R., Heess, N., and Riedmiller, M. (2020). Keep doing what worked: Behavior modelling priors for offline reinforcement learning. In *International Conference on Learning Representations*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354.

Sodhani, S., Zhang, A., and Pineau, J. (2021). Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning*, pages 9767–9779. PMLR.

Srivastava, R. K., Shyam, P., Mutz, F., Jaśkowski, W., and Schmidhuber, J. (2019). Training agents using upside-down reinforcement learning. *arXiv preprint arXiv:1912.02877*.

Stooke, A., Lee, K., Abbeel, P., and Laskin, M. (2020). Decoupling representation learning from reinforcement learning. *arXiv preprint arXiv:2009.08319*.

Stooke, A., Lee, K., Abbeel, P., and Laskin, M. (2021). Decoupling representation learning from reinforcement learning. In *International Conference on Machine Learning*, pages 9870–9879. PMLR.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. (2017). Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind control suite. *arXiv preprint arXiv:1801.00690*.

Todorov, E., Erez, T., and Tassa, Y. (2012a). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033.

Todorov, E., Erez, T., and Tassa, Y. (2012b). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE.

Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., Heess, N., and Tassa, Y. (2020). dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022.

van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., and Modayil, J. (2018). Deep reinforcement learning and the deadly triad. *CoRR*, abs/1812.02648.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Venuto, D., Yang, S., Abbeel, P., Precup, D., Mordatch, I., and Nachum, O. (2022). Multi-environment pretraining enables transfer to action limited datasets. *arXiv preprint arXiv:2211.13337*.

Voloshin, C., Le, H. M., Jiang, N., and Yue, Y. (2019). Empirical study of off-policy policy evaluation for reinforcement learning. *arXiv preprint arXiv:1911.06854*.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2019). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Wang, Q., Xiong, J., Han, L., Liu, H., Zhang, T., et al. (2018). Exponentially weighted imitation learning for batched historical data. In *Advances in Neural Information Processing Systems*, pages 6288–6297.

Wang, R., Du, S. S., Yang, L. F., and Salakhutdinov, R. (2020a). On reward-free reinforcement learning with linear function approximation. *arXiv preprint arXiv:2006.11274*.

Wang, R., Foster, D. P., and Kakade, S. M. (2020b). What are the statistical limits of offline rl with linear function approximation?

Wang, R., Wu, Y., Salakhutdinov, R., and Kakade, S. M. (2021). Instabilities of offline rl with pre-trained neural representation. *arXiv preprint arXiv:2103.04947*.

Wang, Z., Novikov, A., Zolna, K., Merel, J. S., Springenberg, J. T., Reed, S. E., Shahriari, B., Siegel, N., Gulcehre, C., Heess, N., et al. (2020c). Critic regularized regression. *Advances in Neural Information Processing Systems*, 33.

Whitney, W., Agarwal, R., Cho, K., and Gupta, A. (2019). Dynamics-aware embeddings. *arXiv preprint arXiv:1908.09357*.

Wong, A., Zeng, A., Bose, A., Wahid, A., Kalashnikov, D., Krasin, I., Varley, J., Lee, J., Tompson, J., Attarian, M., Florence, P., Baruch, R., Xu, S., Welker, S., Sindhwani, V., Vanhoucke, V., and Gramlich, W. (2022). Pyreach - python client sdk for robot remote control. https://github.com/google-research/pyreach.

Wu, P., Majumdar, A., Stone, K., Lin, Y., Mordatch, I., Abbeel, P., and Rajeswaran, A. (2023). Masked trajectory models for prediction, representation, and control. *arXiv preprint arXiv:2305.02968*.

Wu, Y., Tucker, G., and Nachum, O. (2019). Behavior regularized offline reinforcement learning.

wwPDB consortium (2018). Protein Data Bank: the single global archive for 3D macromolecular structure data. *Nucleic Acids Research*, 47(D1):D520–D528.

Xie, T., Cheng, C.-A., Jiang, N., Mineiro, P., and Agarwal, A. (2021). Bellman-consistent pessimism for offline reinforcement learning. *Advances in neural information processing systems*, 34.

Yalniz, I. Z., Jégou, H., Chen, K., Paluri, M., and Mahajan, D. (2019). Billion-scale semi-supervised learning for image classification. *arXiv preprint arXiv:1905.00546*.

Yang, M., Levine, S., and Nachum, O. (2021). Trail: Near-optimal imitation learning with suboptimal data. *arXiv preprint arXiv:2110.14770*.

Yang, M. and Nachum, O. (2021). Representation matters: offline pretraining for sequential decision making. In *International Conference on Machine Learning*, pages 11784–11794. PMLR.

Yang, M., Schuurmans, D., Abbeel, P., and Nachum, O. (2022). Dichotomy of control: Separating what you can control from what you cannot. *arXiv preprint arXiv:2210.13435*.

Yang, S., Nachum, O., Du, Y., Wei, J., Abbeel, P., and Schuurmans, D. (2023). Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129*.

Yarats, D., Brandfonbrener, D., Liu, H., Laskin, M., Abbeel, P., Lazaric, A., and Pinto, L. (2022). Don't change the algorithm, change the data: Exploratory data for offline reinforcement learning. *arXiv preprint arXiv:2201.13425*.

Yarats, D., Fergus, R., Lazaric, A., and Pinto, L. (2021a). Reinforcement learning with prototypical representations. *arXiv preprint arXiv:2102.11271*.

Yarats, D., Zhang, A., Kostrikov, I., Amos, B., Pineau, J., and Fergus, R. (2021b). Improving sample efficiency in model-free reinforcement learning from images. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10674–10681.

Yu, T., Finn, C., Xie, A., Dasari, S., Zhang, T., Abbeel, P., and Levine, S. (2018). One-shot imitation from observing humans via domain-adaptive meta-learning. *arXiv preprint arXiv:1802.01557*.

Yu, T., Kumar, A., Chebotar, Y., Hausman, K., Levine, S., and Finn, C. (2021). Conservative data sharing for multi-task offline reinforcement learning. *Advances in Neural Information Processing Systems*, 34.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. (2020). Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR.

Zakka, K., Zeng, A., Florence, P., Tompson, J., Bohg, J., and Dwibedi, D. (2022). Xirl: Cross-embodiment inverse reinforcement learning. In *Conference on Robot Learning*, pages 537–546. PMLR.

Zanette, A. (2021). Exponential lower bounds for batch reinforcement learning: Batch rl can be exponentially harder than online rl.

Zang, H., Li, X., Yu, J., Liu, C., Islam, R., Combes, R. T. D., and Laroche, R. (2022). Behavior prior representation learning for offline reinforcement learning. *arXiv preprint arXiv:2211.00863*.

Zhang, A., McAllister, R., Calandra, R., Gal, Y., and Levine, S. (2020). Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*.

Zhang, S. and Jiang, N. (2021). Towards hyperparameter-free policy selection for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 34.

Zhang, T. T., Kang, K., Lee, B. D., Tomlin, C., Levine, S., Tu, S., and Matni, N. (2022). Multi-task imitation learning for linear dynamical systems. *arXiv preprint arXiv:2212.00186*.

Štrupl, M., Faccio, F., Ashley, D. R., Schmidhuber, J., and Srivastava, R. K. (2022). Upside-down reinforcement learning can diverge in stochastic environments with episodic resets.