THE ESSENTIAL ELEMENTS OF OFFLINE RL VIA SUPERVISED LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Offline reinforcement learning (RL) is typically tackled with value-based methods that rely on temporal difference (TD) learning. However, recent work has shown how the offline RL problem can be reduced to weighted, filtered, or conditional behavioral cloning problems. The main idea behind these methods is that data that is suboptimal for one task, such as achieving a certain reward of reaching a certain goal, may nonetheless be useful for learning to solve another task. These methods coerce suboptimal data into a form that allows supervised learning methods to acquire optimal policies. For example, any experience is optimal for learning to reach the final state in that experience. However, it remains unclear which ingredients are essential for such methods to work well, and when these methods do or do not work outperform value-based approximate dynamic programming algorithms. These methods, which we collectively refer to as reinforcement learning via supervised learning (RvS), involve a number of design decisions, such as policy architectures and how the conditioning variable is constructed. Through extensive experiments, this paper studies the importance of these design decisions. The most important design decisions boil down to carefully choosing model capacity (e.g., via regularization or architecture) and choosing which information to condition on (e.g., goals or rewards). Our experiments find that more complex design choices, such as the large sequence models and value-based weighting schemes used in prior work, are often not necessary. Our results show that carefully designed RvS methods can attain results that match or exceed the best prior methods across a range of different offline RL benchmarks, including datasets with little or no optimal data. Our results help to identify the limits of current RvS methods and identify some important open problems.

1 INTRODUCTION

Offline and off-policy reinforcement learning (RL) is typically addressed using value-based methods. While conceptually simple, such methods can be difficult to apply in practice and tend to require a complex tricks to stabilize learning and delicate tuning of many hyperparameters. Recent work has explored an appealing alternative approach: convert the RL problem into a *conditional*, *filtered*, or *weighted* imitation learning problem. Such methods typically use a simple insight: experience that may be suboptimal for one task may be optimal for another task. By conditioning on some piece of information, such as a goal, reward function parameterization, or reward value, such experience can be used for simple behavior cloning (Codevilla et al., 2018; Lynch et al., 2020; Ding et al., 2019; Srivastava et al., 2019; Ghosh et al., 2018; Peng et al., 2019; Neumann et al., 2009; Kumar et al., 2019; Eysenbach et al., 2020; Chen et al., 2021; Janner et al., 2021). We will refer to this set of approaches as RL VIA SUPERVISED LEARNING (RVS). These approaches most commonly conditioning on goals (Lynch et al., 2020; Codevilla et al., 2018; Ghosh et al., 2021) or reward values (Kumar et al., 2019; Srivastava et al., 2019; Chen et al., 2021), though they can also involve reweighting or filtering (Eysenbach et al., 2020; Neumann et al., 2009; Peng et al., 2019; Kumar et al., 2019).

RvS methods are appealing because of their algorithmic simplicity. However, prior works have put forward conflicting hypotheses about which factors are essential for their good performance, including online data (Ghosh et al., 2021), weighting (Kumar et al., 2019), or large Transformer sequence models (Chen et al., 2021). It remains unclear where, how, and why they work well.

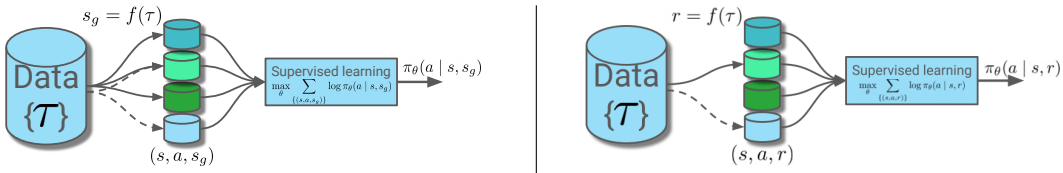
Therefore, the first and most important question we aim to study is: *what elements are essential for effective RvS learning?* Answering this question may help users understand when RvS methods are a generally-applicable replacement for dynamic programming methods. Beyond this, it also remains unclear how or when such methods can effectively exploit temporal compositionality. Dynamic programming methods can make use of this effectively, and it is important for offline RL tasks where few near-optimal trajectories are present in the dataset, as for example in the Ant Maze tasks in D4RL (Fu et al., 2020). Therefore, the second question we aim to answer is: *what are the limits of RvS learning, and does it scale effectively to settings that contain few near-optimal trajectories?*

Our focus is not on proposing new RvS algorithms, but rather on empirically investigating the design decisions, performance, and limits of such methods. Prior works have proposed a number of factors for improving the performance of RvS learning methods, such as filtering or weighting (Kumar et al., 2019), high-coverage "play" data (Lynch et al., 2020), online collection (Ghosh et al., 2021), or the use of large Transformer models (Chen et al., 2021; Janner et al., 2021). Our first result is that, based mostly on careful adjustment of model capacity (e.g., via regularization), simple feedforward models can match the performance of more complex and computationally expensive sequence models from prior work across a wide range of tasks. Our second result is that choosing to condition on reward values versus goals can have a large effect on performance, with different choices working better in different domains.

The main contribution of this paper is a study of RvS methods on a wide range of offline RL problems, as well as a set of analyses about which design factors matter most for such methods. We show that relatively simple network designs can, in many cases, perform as well (or better than) both state-of-the-art offline RL algorithms and more complex architectures, if tuned correctly to have the right capacity. While this may be unsurprising, it contradicts hypotheses put forward in several prior works (Kumar et al., 2019; Chen et al., 2021). Additionally, to our knowledge the results that we demonstrate exceed, on average, previously reported results for any RvS method, when considered across the full range of tasks. We believe that these findings will be useful to the RL community, because they help to understand the essential ingredients and limitations of RvS methods, and provide a foundation for future work on simple and performant offline RL algorithms.

2 Related Work

Recent offline RL methods use many algorithmic techniques, including value functions (Kumar et al., 2020; Fujimoto et al., 2019), dynamics estimation alongisde value functions (Yu et al., 2020; Shen et al., 2021; Kidambi et al., 2020), dynamics estimation alone (Depeweg et al., 2017; Argenson & Dulac-Arnold, 2020; Swazinna et al., 2021), and uncertainty quantification (Agarwal et al., 2020; Wu et al., 2021; Yu et al., 2020). In this paper, we focus on a class of offline RL methods based on conditional behavior cloning that avoid value functions entirely. The most common instantiation of these methods are goal-conditioned behavior cloning (Ghosh et al., 2021; Lynch et al., 2020; Ding et al., 2019) and reward-conditioned behavior cloning (Kumar et al., 2019; Srivastava et al., 2019; Schmidhuber, 2019). Prior work has also looked at conditioning on different information, such as entire trajectories (Chen et al., 2021; Janner et al., 2021), tasks inferred by inverse RL (Eysenbach et al., 2020), and other task information (Codevilla et al., 2018; Hawke et al., 2020). These methods all eschew value estimation and instead solve a supervised regression problem that resembles behavioral cloning. While these methods can, in theory, be directly applied to the offline RL setting, some prior works only evaluate these methods with combined with iterative data collection (Srivastava et al., 2019; Ghosh et al., 2021), which is not permitted in the typical offline RL setting. Kumar et al. (2019) study reward-conditioned policies and do present experiments in the offline RL setting. However, in contrast with our work, the results from Kumar et al. (2019) suggest that good performance cannot be achieved through RvS learning but only through additional advantage-weighting (Neumann et al., 2009; Peng et al., 2019). More recent work (Chen et al., 2021) presents a conditional imitation learning method for the offline RL setting that avoids advantage weighting by introducing a higher capacity, autoregressive Transformer policy. In contrast to these prior works, we find that suitable design decisions in training enable simple conditioning with standard feedforward networks to attain state-of-the-art results across a range of offline RL benchmarks, without advantage weighting or high-capacity sequence models.



(a) RvS-Learning with Goals (RvS-G)

(b) RvS-Learning with Rewards (RvS-R)

Figure 1: RL via Supervised Learning (RvS) learns a goal-conditioned policy by using achieved future goals as supervision. *(Left)* Conditioning on future reached states. *(Right)* Conditioning on cumulative, future achieved reward. RvS can condition on arbitrary goals at both training and inference time.

3 Reinforcement Learning via Supervised Learning

In this section, we will describe a formulation of reinforcement learning via supervised learning. While we will refer to these methods as "RvS learning" methods, we emphasize that we are not proposing any new algorithm, but rather summarizing existing methods under one common framework. In our experiments section, we will discuss the specific design decisions that we find important to make such methods work well. To formulate RvS methods in a general way, we assume an agent interacts with a Markov decision process with states s_t, actions a_t, initial state distribution $p_s(s_1)$, and dynamics $p(s_{t+1} \mid s_t, a_t)$. We assume that the agent chooses actions using a policy $\pi_\theta(a_t \mid s_t)$, parameterized by θ. For simplicity, we assume episodes have fixed length H, and use $\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \cdots)$ to denote a trajectory of experience. As some trajectories might be described using multiple outcomes, we will use $f(\omega \mid \tau)$ to denote the distribution over outcomes that occur in this trajectory. We will study two types of outcomes. The first type of outcome is a state ($\omega \in \mathcal{S}$) that the agent visits in the future: $f(\omega \mid \tau_{t+1:H}) = \text{Unif}(s_{t+1}, s_{t+2}, \cdots, s_H)$. We will refer to RvS methods that learn goal-conditioned policies as *goal-conditioned behavior cloning* (GCBC). The second type of outcome is the average return ($\omega \in \mathbb{R}$) achieved over some future number of time steps: $f(\omega \mid \tau_{t+1:H}) = \mathbb{1}(\omega = \frac{1}{H}\sum_{t'=t+1}^{H+1} r(s_{t'}, a_{t'}))$. We will refer to RvS methods that learn reward-conditioned policies as *reward-conditioned behavior cloning* (RCBC). It is also possible to condition on other information, such as the parameters of a reward function inferred via inverse RL (Eysenbach et al., 2020) or the parameters of a task, such as turning left or right (Codevilla et al., 2018). We will focus on RvS methods applied to the *offline RL* setting. These methods take as input a dataset of experience, $\mathcal{D} = \{\tau\}$, and output an outcome-conditioned policy, $\pi_\theta(a_t \mid s_t, \omega)$, which as in (Ghosh et al., 2021) is the solution to the following optimization problem:

Algorithm 1 RvS-Learning

1: **Input**: Dataset of trajectories, $\mathcal{D} = \{\tau\}$
2: Initialize policy $\pi(a \mid s, \omega)$.
3: **while** not converged **do**
4: Randomly sample trajectories: $\tau \sim \mathcal{D}$.
5: Sample time index for each trajetory, $t \sim [1, H]$, and sample a corresponding outcome: $\omega \sim f(\tau_{t:H})$.
6: Compute loss: $\mathcal{L}(\omega) \leftarrow \sum_{(s_t, a_t, \omega)} \log \pi_\theta(a_t \mid s_t, \omega)$
7: Update policy parameters: $\omega \leftarrow \omega + \eta \nabla_\theta \mathcal{L}(\theta)$
8: **end while**
9: **return** Conditional policy $\pi(a \mid s, \omega)$

$$\max_\theta \frac{1}{|\mathcal{D}|H} \underbrace{\sum_{\tau \in \mathcal{D}}}_{\text{For all trajectories:}} \underbrace{\sum_{1 \le t \le |\tau|}}_{\text{For all states in that trajectory:}} \underbrace{\mathbb{E}_{\omega \sim f(\omega \mid \tau_{t+1:H})}}_{\text{For all achieved outcomes:}} [\log \pi_\theta(a_t \mid s_t, \omega)]. \tag{1}$$

While the basic formulation of this conditional policy is simple, instantiating RvS methods that attain excellent results has proven challenging (Kumar et al., 2019; Srivastava et al., 2019). In the remainder of this paper, we will present an empirical analysis of the design choices in RvS methods. Specifically, we will aim to understand which design choices are needed to make RvS methods perform well on benchmark tasks, including how two different choices for ω (goal and reward values) compare in practice across diverse tasks and datasets with varying levels of optimality (random, mixed, and expert). We will first describe the tasks and datasets that we will use in our

evaluation in Section 4, discuss various design choices in Section 5, and then present comparisons and analysis in Section 6.

4 TASKS AND DATASETS

To provide a detailed and comprehensive empirical study of RvS methods, we selected a broad range of tasks; prior works that proposed various types of RvS or conditional imitation methods typically focused on only a narrow subset of the domains we consider (Kumar et al., 2019; Ghosh et al., 2021; Chen et al., 2021). Our goal will be to include domains and datasets that: *(1)* are appropriate for different types of conditioning, including goals and rewards; *(2)* datasets that include different proportions of near-optimal data, ranging from near-expert datasets to ones with very few or no optimal trajectories at all; *(3)* datasets that run the gamut in terms of task dimensionality; *(4)* domains that have been heavily studied and compared on by prior value-based offline RL methods, thus offering a calibrated point of comparison.

GCSL is a suite of goal-conditioned environments used by Ghosh et al. (2021) to evaluate GCSL, a *goal-conditioned* RvS method with *online* data collection. We adapt these tasks for offline RL by using a *random* policy to collect training data, which results in suboptimal trajectories. The tasks include 2D navigation with obstacles (`FourRooms`, Eysenbach et al. (2019)), two end-effector controlled Sawyer robotic arm tasks (`Door` and `Pusher`, Nair et al. (2018)), the Lunar Lander video game that requires controlling thrusters to land a simulated Lunar Excursion Module (`Lander`), and a task that requires rotating a valve with a robotic claw (`Claw`, Ahn et al. (2019)).

Gym locomotion tasks consist of the `HalfCheetah`, `Hopper`, and `Walker` datasets from the D4RL offline RL benchmark (Fu et al., 2020). We use the `random`, `medium`, `medium-expert`, and `medium-replay` datasets in our evaluations, which consist of (a mixture of different) policies with different levels of suboptimality. Although these tasks are difficult for direct behavior cloning, they can be solved effectively by both RvS and a more naïve filtered BC method.

Franka Kitchen is a 9-DoF robotic manipulation task, paired with a dataset of *human demonstrations*. This task originates from Gupta et al. (2020) and was formalized as an offline RL task in D4RL (Fu et al., 2020). Solving these tasks requires composing multi-step behaviors (e.g., open the microwave, then flip a switch) from component skills. This task includes three datasets: `complete`, which solves all tasks in order, `partial`, where only a subset of trajectories perform the desired tasks in order, and `mixed`, which contains all subtasks but *never* in the right order, requiring generalization and temporal composition. We will see that various methods compare differently on these tasks than on the **Gym locomotion** tasks.

Antmaze involves controlling a 8-DoF quadruped to navigate from a particular goal state. This benchmark task, from D4RL (Fu et al., 2020), evaluates an agent's ability to learn *trajectory stitching* from a *non-Markovian demonstrator policy*. There are three mazes of increasing size: `umaze`, `medium`, and `large`. The data consists of trajectories that traverse between *different* start-goal pairs, with `medium` including few trajectories from the "real" task start to the goal, and `large` including none at all. Prior work has proposed that, with such tasks, dynamic programming with value-based methods should be particularly important (Fu et al., 2020), as extracting near-optimal behavior requires understanding temporal compositionality ("stitching"). We indeed find that these tasks provide an interesting challenge that differs from the other tasks, resulting in different tradeoffs between methods.

When applying RvS learning methods, one must choose a particular type of outcome, such as rewards or goals. Whenever possible, we report numbers for both goal conditioning (RvS-G) and reward conditioning (RvS-R). We note that in Kitchen and AntMaze, we condition on the goal state that defines the underlying reward function (all the subtasks completed in the kitchen, and the goal location in the maze); this makes the additional assumption that we know how the reward function is defined, rather than just receiving samples from the reward function. Because GCSL is inherently a multitask, goal-conditioned setup, we omit reward conditioning and only report RvS-G; conversely, because there is no clear way to define performance in Gym w.r.t. a goal, we omit goal conditioning and only report RvS-R. For all tasks, we report scores in the range $[0, 100]$. For GCSL, the score corresponds to the percentage of episodes in which the agent achieves the goal. In the other environments, we follow Fu et al. (2020) and normalize the score according to $100 \cdot \frac{\text{return} - \text{random}}{\text{expert} - \text{random}}$.

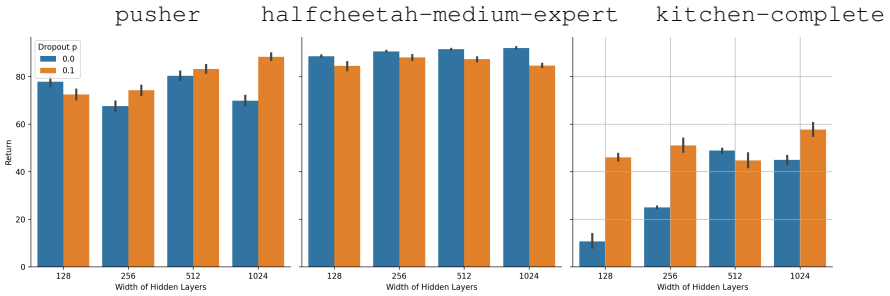


Figure 2: **Network capacity and regularization**: We vary capacity (via network width) and regularization (via dropout) on one validation environment from each task suite. Larger networks perform better on `pusher` and `kitchen-complete`, suggesting the importance of high-capacity policy networks. However, dropout also boosts performance, suggesting that a combination of high-capacity policies with effective regularization is important for achieving good results.

5 Architecture, Capacity, and Regularization

We will instantiate the RvS framework in Section 3 using simple feedforward, fully-connected neural networks, conditioning on either the reward-to-go (Kumar et al., 2019; Srivastava et al., 2019) or a goal state (Lynch et al., 2020; Gupta et al., 2020; Ghosh et al., 2021). Our aim is specifically to identify the *essential* components of such methods, and we will eschew more complex architectures, such as Transformer sequence models (Chen et al., 2021). We will show that on the tasks we consider, we can achieve competitive performance without these components. However, we will show that a number of seemingly minor choices in the architecture design matter a great deal for final performance. Most of these choices revolve around tuning capacity and regularization, suggesting that overfitting and underfitting are major challenges for RvS; we discuss them below. Additionally, we will discuss in Section 6 how the choice of what to condition on (goals and rewards) also has important and domain-specific repercussions.

Capacity and regularization. In Figure 2, we compare different architecture sizes and regularization settings on a single task arbitrarily chosen from each task suite. The best-performing architectures are generally larger than the architectures used in standard online RL (Lillicrap et al., 2015) and imitation learning (Ho & Ermon, 2016; Fu et al., 2017). This conclusion is intuitive, since RvS policies must learn not only the optimal policy *but also the policies for other conditioning values* (e.g., other goals or suboptimal rewards). This result may help explain the contradictory conclusions from prior work regarding the importance of more complex design decisions (Kumar et al., 2019; Chen et al., 2021). The results in Figure 2 also highlight the importance of regularization: in some tasks, like `pusher` and `kitchen-complete`, dropout improves performance, while for others, like `halfcheetah-medium-expert`, dropout worsens performance.

Qualitatively, `kitchen-complete` is the smallest dataset and it contains human demonstrations, which prior work has found is especially hard to fit (relative to artificial demonstrations). On the other hand, `halfcheetah-medium-expert` is a large dataset that (by construction) only has two modes. So it is not surprising that this data is easiest to fit. While `pusher` has lots of data, it is of random trajectories. We conjecture that regularization helps smooth out the randomness in the `pusher` demonstrations. More generally, while our observations about model size and regularization may appear contradictory, we speculate that the issue might come down to the fact that RvS methods must fit a complex action distribution (due to conditioning) while still being regularized well enough to generalize over the goals or reward targets.

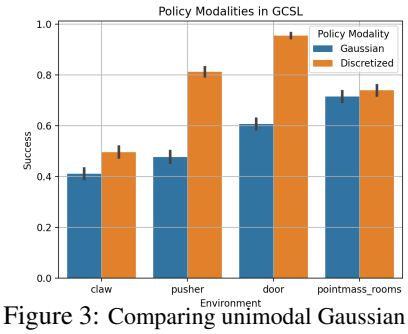


Figure 3: Comparing unimodal Gaussian distributions with (multimodal) discretized distributions. More expressive distributions improve performance.

Output distributions. The form of the policy's output distribution also influences the policy's capacity. With continuous actions, unimodal Gaussians are a common choice,but discretization of

the action space allows representing complex, multi-modal conditional distributions (Lynch et al., 2020; Ghosh et al., 2021). To study the importance of the policy output distribution, we use the low-dimensional GCSL environments, where we can discretize the full action space. Figure 3 shows that discretized output distributions outperform unimodal Gaussian output distributions. These results mirrors those from the previous section in highlighting how increasing the policy's model capacity can improve performance. Again, our finding stands in contrast to standard RL methods, which work well with unimodal Gaussians.We conjecture that this result is caused because RvS policies must represent both optimal and suboptimal behavior (e.g., high-reward targets and low-reward targets), whereas standard RL policies only need to represent optimal behavior. We conjecture that upweighting better data, as down by previous methods (Kumar et al., 2019), may make it easier for low-capacity policies to fit the data. As increasing model capacity is often simpler than learning value functions, our results suggest an easier approach to achieving similar results.

Tuning with validation loss. Hyperparameter optimization is especially important in the offline RL setting, as evaluating different hyperparameter configurations requires interacting with the environment (Paine et al., 2020). Since RvS methods reduce the RL problem to a supervised learning problem, we conjecture that the validation loss of this supervised learning problem might be an effective metric for tuning important hyperparameters, such as policy capacity and regularization. In Figure 4, we train models on 80% of the training data for each Franka Kitchen dataset with two different hyperparameter settings: a regularized setting with dropout $p = 0.1$ and small batch size 256, and an unregularized setting without dropout and a large batch size of $16,384$. While in all three cases validation set error does correlate with performance, the strength of this correlation varies significantly, suggesting that validation loss does not provide a reliable approach for hyperparameter tuning. This suggests that fully automated tuning is an open question. Table 1 summarizes the hyperparameters that we found to work best for each offline RL benchmark.

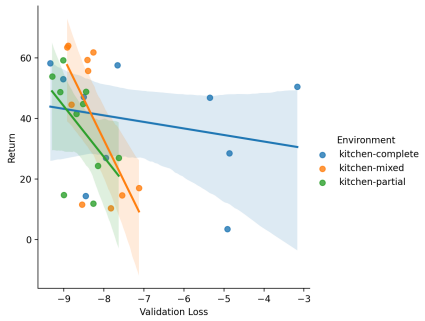


Figure 4: Return vs. validation loss on the `kitchen` environments. While there is sometimes strong correlation, validation loss provides an imperfect gauge on final performance.

A recipe for practicioners. Figure 2 indicates that having enough model capacity is important for strong performance, and regularization sometimes also provides a substantial boost. While offline hyperparameter tuning remains an open challenge (Fu et al., 2021; Paine et al., 2020), we suggest the following process for online hyperparameter tuning: incrementally increase network width until performance saturates, and then try adding a bit of dropout regularization (e.g., dropout $p = 0.1$). If under/overfitting is still an issue, one can also try increasing/decreasing the batch size, respectively.

6 Comparing RvS with Prior Offline RL Methods

Having identified key design decisions in the implementation of RvS learning methods, we now turn to comparing RvS learning methods to prior offlinr RL methods. We will evaluate both goal-conditioned behavioral cloning (RvS-G) and reward-conditioned behavioral cloning (RvS-R) on each of the domains in Section 4, in comparison with relevant baselines and state-of-the-art methods in the literature. We then discuss what these results imply about the performance, design parameters, and limitations of RvS methods. A summary of the results is presented in Figure 5, with detailed task-by-task breakdowns in Appendix A.

Baselines and prior methods. On the D4RL benchmarks, we compare RvS learning to *(i)* value-based methods and *(ii)* prior supervised learning methods and behavioral cloning baselines. For *(i)*, we include CQL (Kumar et al., 2020), with results updated to the v2 D4RL environments for Gym locomotion,[1] as well as the more recent TD3+BC (Fujimoto & Gu, 2021) and Onestep RL (Brand-

[1]Several prior works report results for CQL on the v0 D4RL Gym locomotion environments, which are somewhat harder. We use v2 for all methods for fairness, hence the reported CQL numbers are higher than some prior works.

Table 1: Architecture and design parameters that we found to work best in each domain. We define an epoch length to include all start-goal pairs in GCSL, i.e., to be $|\mathcal{D}|\binom{H}{2}$. In D4RL, we set all epoch lengths at 2000 $\binom{50}{2}$ = 2450000. We use the final checkpoints except in AntMaze, where we pick the best checkpoint based on a 10% validation set.

Hyperparameter	Value	Environment
Hidden layers	2	All
Layer width	1024	All
Nonlinearity	ReLU	All
Learning rate	1e-3	All
Epochs	10	GCSL
	50	Kitchen
	2000	Gym
Gradient steps	20000	AntMaze
Batch size	256	GCSL, Kitchen
	16384	Gym, AntMaze
Dropout	0.1	GCSL, Kitchen
	0	Gym, AntMaze
Goal state	Given	GCSL
	All subtasks completed	Kitchen
	(x, y) location	AntMaze
Reward target	110	Gym medium-expert, AntMaze, Kitchen
	90	Gym {hopper, walker2d}-medium-replay, walker2d-medium
	60	Gym hopper-medium
	40	Gym random, halfcheetah-medium, medium-replay
Policy output	Discrete categorical	GCSL
	Unimodal Gaussian	Kitchen, Gym, AntMaze

fonbrener et al., 2021) methods, which involve elements of behavioral cloning as well. For *(ii)*, we include behavioral cloning (BC), which does not perform any conditioning; "10% BC," which imitates the best 10% of the trajectories; and Decision Transformers (DT) (Chen et al., 2021), which conditions on rewards and uses a large Transformer sequence model. The DT baseline used code released by the authors, which was tuned on Antmaze based on correspondence with the authors; both BC and 10% BC are our own optimized implementations, tuned thoroughly in a similar manner as RvS-R and RvS-G. Due to time constraints, we were unable to adequately tune DT, TD3+BC, and Onestep RL on Kitchen (these numbers are also not available from the authors). On the GCSL benchmarks, we compare to GCSL (Ghosh et al., 2021) using numbers reported by the authors, which use online data. We do not run RvS-G on the Gym locomotion tasks, which are typically expressed as reward-maximization tasks, not goal-reaching tasks.

Overall performance and comparison to prior methods. The results in Figure 5 lead to several conclusions about how our RvS implementations compare to prior value-based and supervised methods. First, on the GCSL, Kitchen, and Gym Locomotion tasks, either RvS-G or RvS-R attains results that are comparable to the best prior method. We will discuss the particular tradeoffs between goals and rewards later. The finding that conditional imitation with standard fully connected networks attains competitive results stands in contrast to conclusions drawn in prior work. As we discussed in Section 5, good performance requires a counterintuitive combination of high capacity and regularization, and it may simply be that this seemingly contradictory combination of components was not adequately explored in prior work. These results suggest that complex sequence model architectures are likely not necessary for these tasks, and the benefits reported by Chen et al. (2021) are likely derived from careful capacity tuning (as we discuss in Section 5) rather than the use of a Transformer.

Subtrajectory stitching. The mixed dataset in Kitchen and all of the Antmaze tasks consist of highly suboptimal trajectories, so expert performance requires recombining parts of these trajectories to attain more optimal behavior. As subtrajectory stitching has usually been viewed as a key benefit of dynamic programming methods, it is not surprising that those baselines that perform dynamic programming (TD3+BC and CQL) perform better than other baselines on this task. In both domains, RvS-G attains relatively good results, matching or outperforming all methods except CQL.

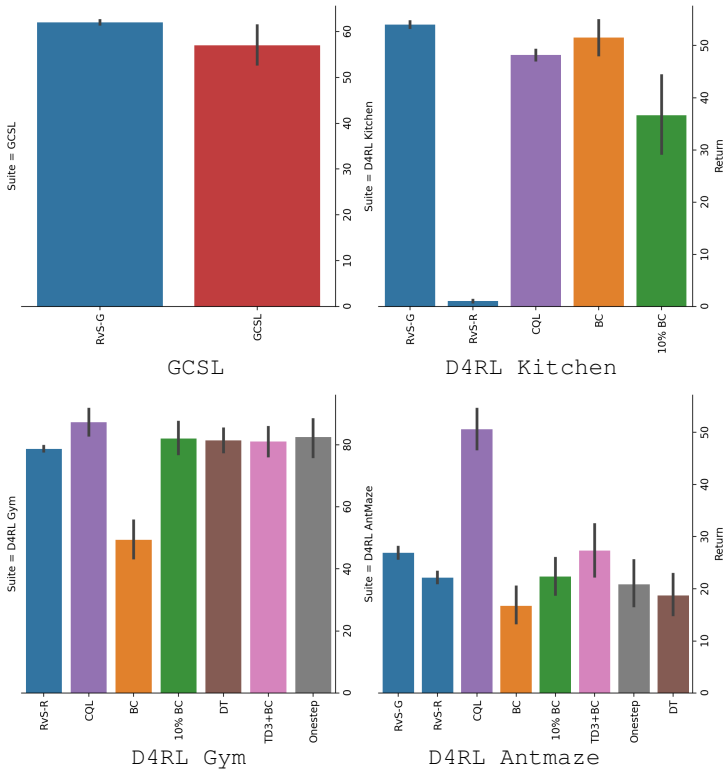
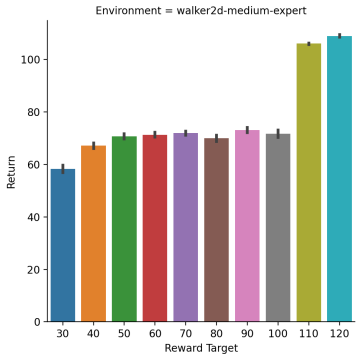


Figure 5: Comparisons between goal-conditioned (RvS-G) and reward-conditioned (RvS-R) methods, as well as prior approaches and baselines. RvS results are averaged over 5 random training seeds, 200 rollouts, and multiple tasks. See Appendix A for more detailed breakdowns. See Appendix Fig. 7 for a larger version.

We believe that this observation supports our discussion at the end of Section 4: the choice of conditioning information for RvS communicates prior knowledge, and this prior knowledge can aid in generalization. In this case, conditioning on goals, which are spatially structured, may provide the method with a similar kind of compositionality in *space* as Bellman backups provide in *time* for dynamic programming techniques. Note, however, that this observation is likely domain-specific, as we discuss later. BC works well in the Kitchen environment, which we speculate is due to the fact that the actual number of skills is low, so simply imitating all available skills can complete the relevant subtasks with sufficient frequency. This is not the case for Antmaze.

Reaching arbitrary goals. An important capability of RvS-G, compared with more standard offline RL methods, is that it learns a policy that can reach many goals. To test how well RvS-G can learn to reach arbitrary goals from random offline data, we compare to GCSL, an RvS learning method that performs *iterative* online data collection (Ghosh et al., 2021). We use the same tasks used by GCSL. Each of the tasks in the GCSL suite requires that the agent reach a goal drawn uniformly at random from the set of possible goals. The results shown in Figure 5 indicate that our policy can successfully reach many different goals entirely from randomly collected offline data. Although it may still be that iterative online collection, as proposed in prior work (Ghosh et al., 2021; Srivastava et al., 2019), may be helpful in some domains, in this case simple offline training appears to suffice.

Random datasets. While RvS-R is generally competitive across the Gym Locomotion tasks, TD learning approaches generally have an edge of a few points in the random datasets, with CQL performing especially well in `halfcheetah-random`. This generally aligns with our expectations, since random datasets provide the most potential for TD methods to improve over conditional behavioral cloning methods. Indeed, we find that the other behavioral cloning methods we consider,



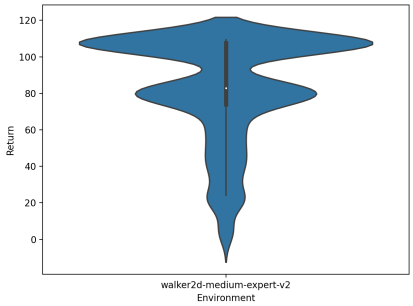


Figure 6: (Left) Varying the reward target in `walker2d-medium-expert`. (Right) The distribution of demonstrator reward in `walker2d-medium-expert`. We see that RvS-R captures the two reward modes in the demonstrator data but does not generalize to arbitrary reward.

BC and %BC, also have a relatively more difficult time with the random datasets compared to the other dataset types.

Analysis of reward conditioning. Next, we further analyze what RvS-R actually learns, and how it uses the conditioning variable. First, we use the `walker2d-medium-expert` task to analyze the behavior of RvS-R for different conditioning rewards. This dataset contains two modes ("medium" and "expert", as shown in Figure 6 *(right)*, and although we can condition the network on *any* reward value, we see in Figure 6 *(left)* that the policy actually learns to achieve just the two modes in the dataset; it cannot interpolate between them. In effect, RvS is extracting a subset of the data, but doing so without explicit filtering (as is done in 10% BC).

This analysis highlights the importance of the conditioning variable. It is not obvious *a priori* that a reward target of 100 will lead to a return of 70, whereas a reward target of 110 will lead to a return of 105. For an apples-to-apples comparison with prior work (Chen et al., 2021), which includes the reward target as a hyperparameter that is tuned for best online performance, we also vary the reward target as a hyperparameter, using the settings in Table 1. However, we believe that this suggests that further investigation into the importance of the conditioning variable, as well as ways to choose hyperparameters in a truly offline setting, is an important problem for future work.

7 Discussion and Future Work

We presented a detailed empirical study of offline RL via supervised learning (RvS) methods, which solve RL problems via conditional imitation learning (Codevilla et al., 2018; Lynch et al., 2020; Ghosh et al., 2021; Eysenbach et al., 2020; Kumar et al., 2019; Srivastava et al., 2019; Chen et al., 2021). While prior work is divided on which particular elements of RvS are crucial for good performance, or even how well such methods actually work, our results provide a number of potentially surprising conclusions. *First*, RvS methods can work well, even with simple fully connected architectures, if capacity and regularization are chosen correctly. *Second*, RvS methods can often attain results that compare favorably to the best prior methods, but only if the conditioning variable (e.g., goal or reward) is chosen appropriately. *Third*, RvS methods can enjoy some of the benefits of compositionality ("stitching"), but only for an appropriate choice of conditioning variable. These conclusions imply a number of interesting directions for future work, which we encourage the community to explore. First, as we discuss in Section 5, validation error does not provide a fully reliable metric for tuning capacity, and because capacity and regularization are so important, appropriate offline evaluation techniques may be essential to make full use of RvS for offline RL. Second, as we showed in Section 6, conditioning is important, and developing more general or more automated ways to select *what* to condition could likely further improve the generality of such methods. Taken together, we believe that our analysis provides clarity on RvS learning methods and also presents a number of avenues for increasing the performance of and further simplifying RvS learning methods.

REFERENCES

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, 2020.

Michael Ahn, Henry Zhu, Kristian Hartikainen, Hugo Ponte, Abhishek Gupta, Sergey Levine, and Vikash Kumar. ROBEL: RObotics BEnchmarks for Learning with low-cost robots. In *Conference on Robot Learning (CoRL)*, 2019.

Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning. *arXiv preprint arXiv:2008.05556*, 2020.

David Brandfonbrener, William F Whitney, Rajesh Ranganath, and Joan Bruna. Offline RL without off-policy evaluation. *arXiv preprint arXiv:2106.08909*, 2021.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *CoRR*, abs/2106.01345, 2021. URL `https://arxiv.org/abs/2106.01345`.

Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4693–4700. IEEE, 2018.

Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and policy search in stochastic dynamical systems with bayesian neural networks. In *International Conference on Learning Representations (ICLR 2017)*, 2017.

Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. In *Advances in Neural Information Processing Systems*, pp. 15324–15335, 2019.

Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 15246–15257, 2019.

Benjamin Eysenbach, Xinyang Geng, Sergey Levine, and Ruslan Salakhutdinov. Rewriting history with inverse RL: hindsight inference for policy improvement. In *Advances in Neural Information Processing Systems*, 2020.

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Justin Fu, Mohammad Norouzi, Ofir Nachum, George Tucker, ziyu wang, Alexander Novikov, Mengjiao Yang, Michael R Zhang, Yutian Chen, Aviral Kumar, Cosmin Paduraru, Sergey Levine, and Thomas Paine. Benchmarks for deep off-policy evaluation. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=kWSeGEeHvF8`.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2106.06860*, 2021.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pp. 2052–2062. PMLR, 2019.

Dibya Ghosh, Abhishek Gupta, and Sergey Levine. Learning actionable representations with goal conditioned policies. In *International Conference on Learning Representations*, 2018.

Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Manon Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=rALA0Xo6yNJ`.

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Conference on Robot Learning*, pp. 1025–1037, 2020.

Jeffrey Hawke, Richard Shen, Corina Gurau, Siddharth Sharma, Daniele Reda, Nikolay Nikolov, Przemysław Mazur, Sean Micklethwaite, Nicolas Griffiths, Amar Shah, et al. Urban driving with conditional imitation learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 251–257. IEEE, 2020.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29:4565–4573, 2016.

Michael Janner, Qiyang Li, and Sergey Levine. Reinforcement learning as one big sequence modeling problem. *arXiv preprint arXiv:2106.02039*, 2021.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. MOReL: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*, 2020.

Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*, 2019.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems*, 2020.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on Robot Learning*, pp. 1113–1132. PMLR, 2020.

Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *Advances in Neural Information Processing Systems*, 31:9191–9200, 2018.

Gerhard Neumann, Jan Peters, et al. Fitted Q-iteration by advantage weighted regression. In *Advances in Neural Information Processing Systems 21-Proceedings of the 2008 Conference*, pp. 1177–1184, 2009.

Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*, 2020.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

Jian Shen, Mingcheng Chen, Zhicheng Zhang, Zhengyu Yang, Weinan Zhang, and Yong Yu. Model-based offline policy optimization with distribution correcting regularization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 174–189. Springer, 2021.

Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz, Wojciech Jaśkowski, and Jürgen Schmidhuber. Training agents using upside-down reinforcement learning. *arXiv preprint arXiv:1912.02877*, 2019.

Phillip Swazinna, Steffen Udluft, and Thomas Runkler. Overcoming model bias for robust offline deep reinforcement learning. *Engineering Applications of Artificial Intelligence*, 104:104366, 2021.

Yue Wu, Shuangfei Zhai, Nitish Srivastava, Joshua Susskind, Jian Zhang, Ruslan Salakhutdinov, and Hanlin Goh. Uncertainty weighted actor-critic for offline reinforcement learning. *arXiv preprint arXiv:2105.08140*, 2021.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. MOPO: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

A Performance Breakdown for Each Task

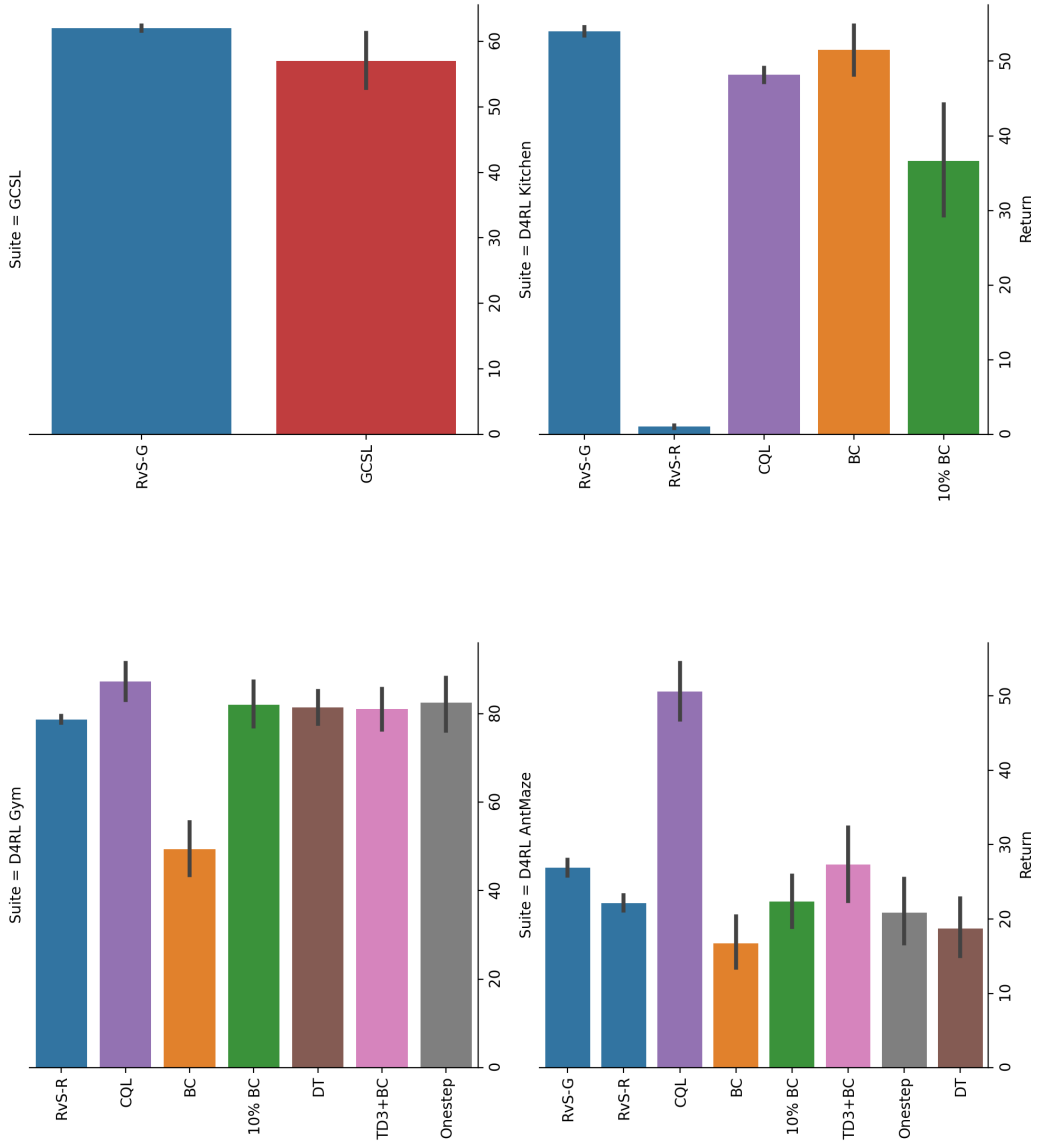
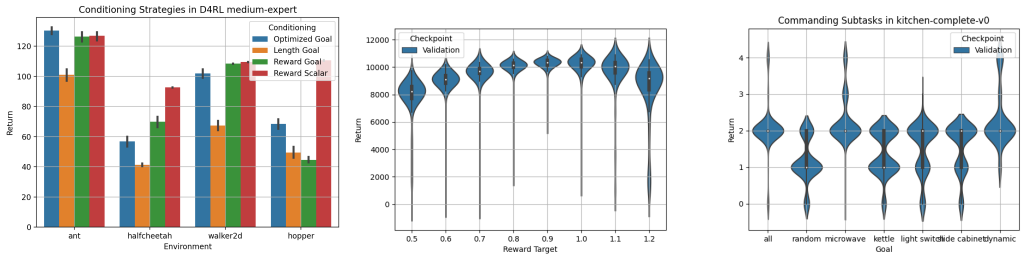


Figure 7: Same plot as Fig. 5, but enlarged to improve readability. Comparisons between goal-conditioned (RvS-G) and reward-conditioned (RvS-R) methods, as well as prior approaches and baselines. RvS results are averaged over 5 random training seeds, 200 rollouts, and multiple tasks.

Table 2: Detailed performance breakdown for every algorithm in every environment.

Suite	Algorithm Environment	10% BC	BC	CQL	DT	GCSL	Onestep	RvS-G	RvS-R	TD3+BC
D4RL AntMaze	large-diverse	6.0	0.0	14.9	0.0		0.0	5.0	3.4	0.0
	large-play	0.0	0.0	15.8	0.0		0.0	3.8	5.9	0.2
	medium-diverse	9.8	0.0	53.7	0.0		0.0	24.9	7.2	3.0
	medium-play	5.4	0.0	61.2	0.0		0.3	9.5	3.5	10.6
	umaze	62.8	54.6	74.0	59.2		64.3	59.1	55.2	78.6
	umaze-diverse	50.2	45.6	84.0	53.0		60.7	59.1	57.8	71.4
D4RL Gym	halfcheetah-medium	42.5	42.6	44.0	42.6		48.4		41.6	48.3
	halfcheetah-medium-expert	92.9	55.2	91.6	86.8		93.4		92.2	90.7
	halfcheetah-medium-replay	40.6	36.6	45.5	36.6		38.1		38.0	44.6
	halfcheetah-random	2.0	2.3	29.7			3.2		3.9	11.0
	hopper-medium	56.9	52.9	58.5	67.6		59.6		60.2	59.3
	hopper-medium-expert	110.9	52.5	105.4	107.6		103.3		101.7	98.0
	hopper-medium-replay	75.9	18.1	95.0	82.7		97.5		73.5	60.9
	hopper-random	4.1	4.8	8.9			7.5		7.7	8.5
	walker2d-medium	75.0	75.3	72.5	74.0		81.8		71.7	83.7
	walker2d-medium-expert	109.0	107.5	108.8	108.1		113.0		106.0	110.1
	walker2d-medium-replay	62.5	26.0	77.2	66.6		49.5		60.6	81.8
	walker2d-random	1.7	1.7	6.7			5.6		-0.2	1.6
D4RL Kitchen	kitchen-complete	4.0	65.0	43.8				50.2	1.5	
	kitchen-mixed	40.0	51.5	51.0				60.3	1.1	
	kitchen-partial	66.0	38.0	49.8				51.4	0.5	
GCSL	claw					28.0		14.8		
	door					27.0		95.8		
	lunar					70.0		54.7		
	pointmass_rooms					80.0		63.0		
	pusher					80.0		81.8		



(a) D4RL Gym (b) `halfcheetah-med-expert` (c) `kitchen-complete`

Figure 8: (a) Different strategies for choosing goals in D4RL Gym. (b) Varying the fraction of expert reward used as a reward target in `halfcheetah-medium-expert`. (c) Commanding the policy to complete different subtasks in `kitchen-complete`.

B Comparison of Conditioning Strategies

To further study how different goal and reward conditioning strategies comparison, in Figure 8a, we show the performance of four strategies: `Reward Scalar`, `Reward Goal`, `Length Goal`, and `Optimized Goal`. In Reward Scalar, we condition RCBC on various fractions of expert performance. In Reward Goal, we randomly select timesteps near the end of the highest-reward demonstration trajectories. In Length Goal, we randomly select timesteps near the end of the longest demonstration trajectories. In Optimized Goal, we do a random search over 200 Length Goals and select the highest performing one. Note that these selection strategies require access to the environment, so they are not strictly offline RL. We nevertheless include them here as ablations to provide additional insight into the goal-conditioned policies. Note that, although we can utilize goals with oracle goal selection strategies for the Gym tasks, the performance of these strategies generally does not exceed reward conditioning, since the tasks are inherently not goal-conditioned.