# 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 TD methods. However, a number of approaches have been proposed that aim to simplify offline RL by reducing it to weighted, filtered, or conditional behavioral cloning problems. 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 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 some prior works, are generally 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, which learn some sort of value function or Q-function. While conceptually simple, such methods can be difficult to apply in practice, and tend to incorporate a number of complex tricks and to stabilize learning and require tuning a number of delicate hyperparameters. Recent work has explored an appealing alternative approach, converting 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 parameter, 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). Most commonly, this consists of 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 it 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 particularly 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 answer is: *what are the essential elements of effective RvS learning?* Are RvS methods a generally-applicable replacement for state-of-the-art 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. Perhaps in contradiction with prior work, our results suggest that the performance of RvS algorithms depends mostly on careful training of basic feedforward neural networks. Though prior works have suggested that performance can be improved using a variety of factors, 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), we find that the main factor determining good performance is carefully adjusting the capacity of the policy (e.g., via regularization). Perhaps surprisingly, this allows very simple feedforward models to match or outperform more complex and computationally expensive sequence models. However, we do observe that one design decision is important in some of the more complex domains that have comparatively little near-optimal data available: the choice of whether the policy should be conditioned on goals or reward values.

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 finds 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

Previously proposed offline RL methods make use of a number of tools, including learning value functions (Kumar et al., 2020; Fujimoto et al., 2019), dynamics estimation (Yu et al., 2020; Shen et al., 2021; Argenson & Dulac-Arnold, 2020; Kidambi et al., 2020), 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 (usually) avoid the use of 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 directly solve some supervised regression problem, which resembles behavioral cloning. Srivastava et al. (2019) and Ghosh et al. (2021) propose frameworks for RL based around conditional imitation learning, proposing that online data collection with iterative relabeling are important building blocks, and do not evaluate on offline RL problems. 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 conditional imitation learning alone is insufficient, and that additional advantage-weighting (Neumann et al., 2009; Peng et al., 2019) is important for good performance. More recent work Chen et al. (2021) presents a conditional imitation learning method 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.

## 3 Reinforcement Learning via Supervised Learning

In this section, we will describe a general formulation of reinforcement learning via supervised learning, which we refer to as RvS for brevity as a way to refer to the entire class of such methods. Note that we are not proposing any new algorithm, but rather summarizing existing methods under

one common framework. However, 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 $s_t$, 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 $\theta$. 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. A given trajectory can be described in terms of one or multiple outcomes, $\omega \in \Omega$. We will use $f(\tau) \subseteq \Omega$ to denote the set of outcomes that apply to a particular trajectory $\tau$. We will study two types of outcomes. The first type of outcome is a state that the agent visits in the future: $f(\tau_{t+1:H}) = \{s_{t+1}, s_{t+2}, \cdots, s_H\}$. 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 achieved over some future number of time steps: $f(\tau_{t+1:H}) = \{\frac{1}{H-t}\sum_{t'=t+1}^{H} r(s_{t'}, a_{t'})\}$. 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). (Offline) RvS methods take as input a dataset of experience, $\mathcal{D} = \{\tau\}$, and output an outcome-conditioned policy, $\pi_\theta(a_t \mid s_t, \omega)$. This policy is the solution to the following optimization problem:

$$\max_\theta \sum_{\substack{\tau \in \mathcal{D}, \\ 1 \leq t \leq H-1, \\ \omega \in f(\tau_{t+1:H})}} \frac{1}{|\mathcal{D}| \cdot H \cdot |f(\tau_{t+1:H})|} \log \pi_\theta(a_t \mid s_t, \omega). \tag{1}$$

While the basic formulation of this conditional policy is remarkably simple, instantiating RvS methods that actually attain excellent results has proven quite challenging (Kumar et al., 2019; Srivastava et al., 2019). In the remainder of this paper, we will present a detailed empirical analysis of the design choices in RvS methods. Specifically, we will aim to understand which design choices are needed to make them perform well on a broad range of prior benchmark tasks, how two different choices for $\omega$ (goal and reward values) compare in practice, and how the nature of the tasks and datasets impacts their performance in offline RL settings. We will first describe the tasks and datasets that we will use in our evaluation in Section 4, then 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 particularly broad range of tasks proposed in various prior papers. Prior works that proposed various types of RvS or conditional imitation methods typically focused on narrower (and often disjoint) task domains. 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 an informative and calibrated point of comparison.

**GCSL** is a suite of goal-conditioned environments utilized by Ghosh et al. (2021) specifically 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 naïve, 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 LEM (`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 `medium-expert` and `medium-replay` datasets in our evaluations, which consist of a mixture of different policies with different levels of suboptimality. As we will see, while these tasks are difficult for direct imitation, they can be solved effectively by both RvS and a more naïve filtered BC method.

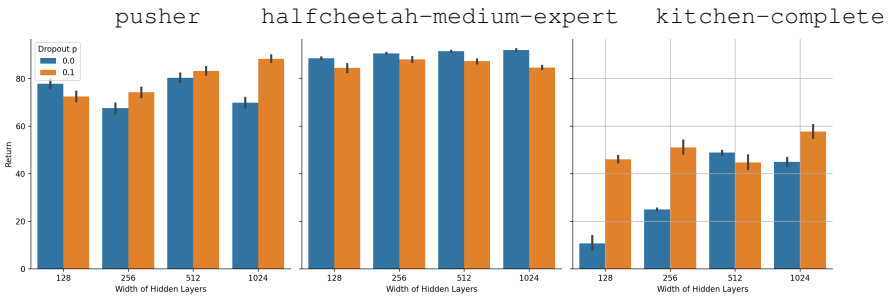



Figure 1: **Network capacity and regularization**: We vary capacity (via network width) and regularization (via dropout) on one validation environment from each task suite. Larger networks perform better on `pusher` and `kitchen-complete`, but surprisingly, dropout boosts performance considerably on these same tasks.

**Franka Kitchen** consists of robotic manipulation datasets, with 9-DoF joint control and *human demonstrations*, that require composing multi-step behaviors (e.g., open the microwave, then flip a switch) from component skills, based on a task from Gupta et al. (2020) and formalized in D4RL (Fu et al., 2020). It 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 very differently on these tasks than on **Gym locomotion**.

**Antmaze**, from D4RL (Fu et al., 2020), evaluates an agent's ability to learn *trajectory stitching* from a *non-Markovian demonstrator policy*, in order to control an 8-DoF quadruped to walk from a start to a goal. 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.

For goal-reaching tasks, such as those in GCSL, we will apply goal-conditioned behavioral cloning (RvS-G) and set the commanded goal to be the goal state. Other tasks, such as Kitchen or Antmaze, are ostensibly reward-driven, but can also be expressed as reaching a single goal state. Note, however, that this does imply an additional assumption on prior knowledge, both in knowing that goal-conditioned RvS methods can be used, and in knowing *which* goal to command. For some tasks, such as the Gym locomotion tasks, goal conditioning does not make sense, and we will only evaluate reward conditioning (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 all other environments, we follow Fu et al. (2020) and normalize the score according to $100 \cdot (\text{return} - \text{random})/(\text{expert} - \text{random})$.

## 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, they do not lead to improvement. However, we will show that a number of seemingly minor choices in the architecture design *do* matter a great deal for final performance. Most of these choices revolve around trading off capacity and regularization, suggesting that overfitting and underfitting are major challenges for RvS. 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.** If Figure 1, we present a comparison of different architecture sizes and regularization settings on a few validation tasks chosen from each task suite for tuning. Note that the best-performing architectures are generally quite large, as compared to architectures using in standard online RL (Lillicrap et al., 2015) and imitation learning (Ho & Ermon, 2016; Fu et al.,

2017). While 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), we believe it may help explain contradictory conclusion in prior work, which found such methods to perform poorly (Kumar et al., 2019) or require much more complex architectures (Chen et al., 2021). Less obvious is the observation that regularization plays a delicate role: in some tasks, like `pusher` and `kitchen-complete`, it improves performance (sometimes considerably), while for others, like `halfcheetah-medium-expert`, it hinders it.

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 — speculatively — 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 – or even extrapolate – over the goal or reward.

**Output distributions.** Policy capacity is also influenced by the form of the output distribution. With continuous actions, unimodal Gaussians are the most common choice, but discretization of the action space allows representing complex, multi-modal conditional distributions. Here, we use the low-dimensional GCSL environments, where we can discretize the full action space, though per-dimension discretizations could be used with higher-dimensional action outputs (Lynch et al., 2020). Figure 2 shows that the more expressive distribution indeed improves performance. This agrees with our previous observation that, properly regularized, greater capacity improves performance. We note again that this stands in contrast to standard RL methods, which work well with unimodal Gaussians. Since RvS policies aim to represent policies for both optimal and suboptimal conditioning values (goals and rewards), insufficient capacity that leads to high error for suboptimal data, which is likely to be more multimodal, can in result in the model not dedicating sufficient capacity to the optimal data. This may also explain the finding in prior work that up-weighting better data can improve performance (Kumar et al., 2019), and conjecture that this may be due to insufficient capacity.

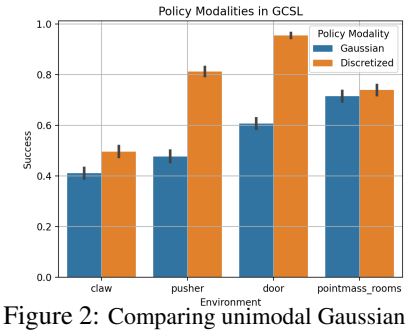



Figure 2: Comparing unimodal Gaussian distributions with (multimodal) discretized distributions. More expressive distributions improve performance.

**Tuning with validation loss.** Since RvS methods are sensitive to capacity and regularization, we might wonder whether we can tune these values offline using a validation set. In Figure 3, 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 this does not provide a fully reliable approach for hyperparameter tuning. This suggests that fully automated tuning is an open question. Following prior work (Fu et al., 2020), we tuned our methods using validation tasks in each domain.

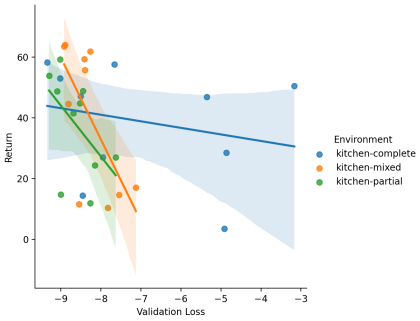



Figure 3: Return vs. validation loss on the `kitchen` environments. While there is sometimes strong correlation, validation loss provides an imperfect gauge on final performance.

**Proposed architecture and parameters.** Due to space constraints, we cannot present full ablations of every architecture parameter, and instead sought to present the parameters that most clearly influence capacity (size and regularization). For completeness, we summarize the specific settings that we ended up using in each domain below.

Table 1: Architecture and design parameters that we found to work best in each domain.

| Hyperparameter | Value | Environment |
|---|---|---|
| Hidden layers | 2 | All |
| Layer width | 1024 | All |
| Nonlinearity | ReLU | All |
| Dropout | 0.1 | GCSL, Kitchen |
| | 0 | Gym, AntMaze |
| Batch size | 256 | GCSL, Kitchen |
| | 16384 | Gym, AntMaze |
| Goal state | Given | GCSL |
| | All subtasks completed | Kitchen |
| | (x, y) location | AntMaze |
| Reward target | 110 | Gym medium-expert, AntMaze, Kitchen |
| | 40 | Gym halfcheetah-medium-replay |
| | 90 | Gym {hopper, walker2d}medium-replay |
| Policy output | Discrete categorical | GCSL |
| | Unimodal Gaussian | Kitchen, Gym, AntMaze |

## 6 Experimental Results

In our results, 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 4, with detailed task-by-task breakdowns in Appendix A.

**Baselines and prior methods.** On the D4RL benchmarks, we compare 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 (Brandfonbrener et al., 2021) methods, which involve elements of behavioral cloning as well. For (ii), we include naïve behavioral cloning (BC) without 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. DT used author code and was tuned on Antmaze based on correspondence with the authors, and 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. Note that we cannot run RvS-G on the Gym locomotion tasks, since these tasks cannot be defined by individual goals. Though conceivably it may be possible to parameterize these tasks in some other way (e.g., in terms of a goal velocity), this involves introducing more detailed per-task prior knowledge.

**Overall performance and comparison to prior methods.** The results in Figure 4 lead to several conclusions about how our RvS implementations compare to prior value-based and supervised methods. First, note that either RvS-G or RvS-R attains results that are comparable to the best prior method in each domain, with the exception of Antmaze. We will discuss the particular tradeoffs between goals and rewards later. The results show that, on the Gym locomotion tasks, RvS-R generally matches or exceeds the performance of BC, 10% BC, and DT. The latter in particular suggests that complex sequence model architectures, with correspondingly lavish training times, 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. However, the good performance of 10% BC also suggests that reward conditioning itself likely attains comparatively less benefit. Overall, it may seem surprising that conditional imitation with standard fully connected networks attains such competitive results, perhaps in contrast to conclusions drawn in prior work. As we discussed in Section 5, good performance requires a counterintuitive combi-

[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.

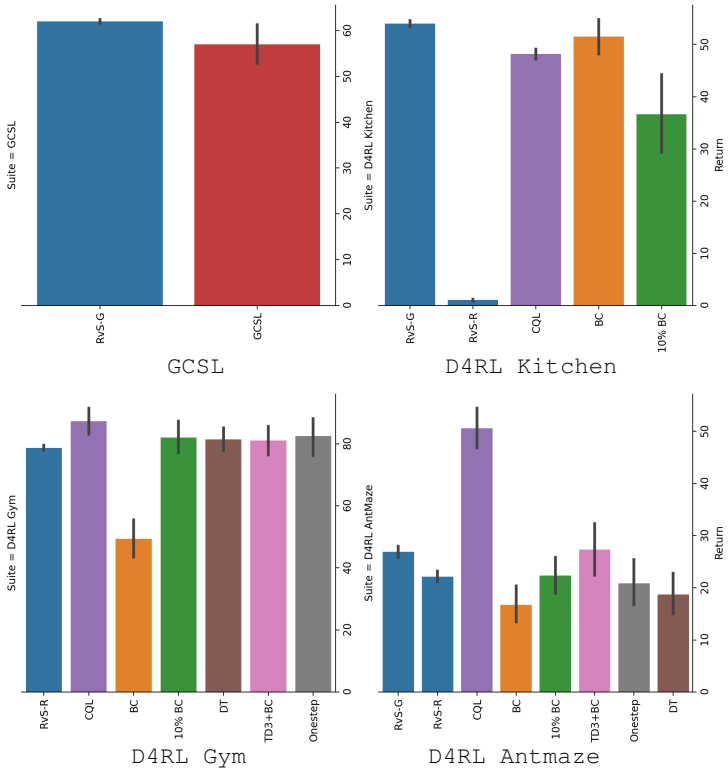
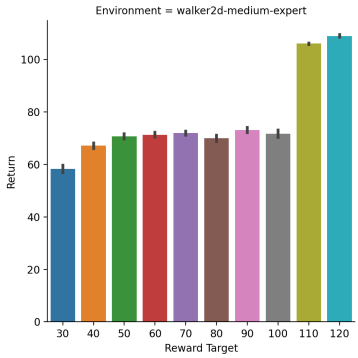



Figure 4: Comparisons between goal-conditioned (RvS-G) and reward-conditioned (RvS-R) methods, as well as prior approaches and baselines. RvS results are averaged over 5 random training seeds, 200 rollouts, and multiple tasks. See Appendix A for more detailed breakdowns. See Appendix. Fig. 6 for a larger version.

nation of high capacity and regularization, and it may simply be that this seemingly contradictory combination of components was not adequately explored in prior work.

**Subtrajectory stitching.** Both the Kitchen and Antmaze tasks consist of highly suboptimal trajectories, and require recombining parts of these trajectories to attain more optimal behavior, with Antmaze being particularly pronounced in this regard. Antmaze is also the task where CQL, which performs explicit dynamic programming, attains by far the largest advantage over other methods. Interestingly, we observe that in both domains, RvS-G attains relatively good results, matching or outperforming all methods except CQL (but including TD3+BC and Oneshot RL, which are value-based methods that use dynamic programming) while RvS-R does not. 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, provides 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 likely completes the full task with sufficient frequency. This is not the case for Antmaze.

**Reaching arbitrary goals.** To test how well RvS-G can learn to reach arbitrary goals from random offline data, we also compare to GCSL, which also uses supervised learning, but performs *iterative* online data collection (Ghosh et al., 2021), using the same tasks as used by GCSL. Each of the tasks in the GCSL suite requires the agent to reach a goal drawn uniformly at random from the set of possible goals. That our method is competitive with GCSL indicates 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.



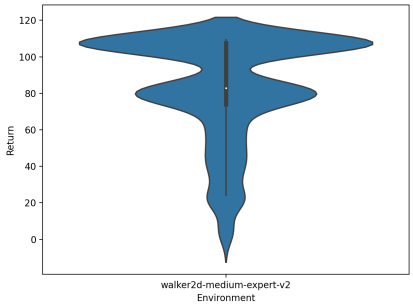




Figure 5: (Left) Varying the reward target in `walker2d-medium-expert`. (Right) The distribution of demonstrator reward in `walker2d-medium-expert`. We see that RvS-R captures the two reward modes in the demonstrator data but does not generalize to arbitrary reward.

**What to condition on?** Based on the above discussion, we might conclude that RvS-G provides a better method than RvS-R. However, we would note that in some cases, it cannot be used effectively, such as for the Gym locomotion tasks, which lack a single clear goal. In fact, we can interpret RvS-G and its good performance as resulting from a subtle incorporation of domain knowledge that some tasks have goals, and the goal is known. Note that *analogous* knowledge could likely be devised for other tasks: for example, we might conditioning the Gym locomotion tasks on a target velocity, or some other task-relevant variable. Viewed comparatively with value-based methods (e.g., CQL, TD3+BC), we might also say that, while value-based methods derive compositionality from the structure of the TD backup, RvS-G (and other types of conditioning) derive compositionality from *structure in the conditioning variable* (e.g., the spatial structure in goals). This implies that good performance for these methods does hinge to a considerable degree on a judicious choice of conditioning variable, and this conclusion is further supported by the comparatively worse performance of the (more general) RvS-R variant. This provides us with a conclusion regarding the *limitations* of RvS: since such methods lack the temporal compositionality that is inherent in value-based methods, they must *recover* this compositionality by some other means, such as structure in the conditioning variable, and they must have the right balance of capacity and regularization to utilize this structure appropriately, which typically means larger capacity than is typical in standard RL (see discussion in Section 5). We provide additional analysis comparing different types of reward and goal conditioning strategies in Appendix B, along with some analysis of how RvS-G could be used on the Gym tasks with oracle goal selection.

**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 5(right)), and although we can condition the network on *any* reward value, we see in Figure 5(left) that the policy actually learns to achieve just the two modes in the dataset. It does not generalize beyond these modes, and actually does not interpolate between them. In effect, RvS is extracting a subset of the data, but doing so more flexibly than 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 use a reward target of 110 for all the `medium-expert` datasets, a reward target of 40 for `halfcheetah-medium-replay`, and a reward target of 90 for the remaining `medium-replay` datasets. 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, focusing on offline RL, analysis RL via supervised learning (RvS) methods, which refers to a class of methods explored in multiple recent works (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) that propose to solve RL problems via some sort of conditional imitation learning. While prior work is generally 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: (1) RvS methods can work well even with simple fully connected architectures, but only if capacity and regularization are chosen correctly; (2) RvS methods can often attain results that comparable favorably to the best prior methods, but only if the conditioning variable (e.g., goal or reward) is chosen appropriately; (3) RvS methods can enjoy some of the benefits of compositionality ("stitching"), but again only for an appropriate choice of conditioning variable, as it is the structure in this conditioning variable that provides them with a suitably compositional space to generalize over, and generally do not perfom as well as the best value-based methods (i.e., CQL), though are competitive with some value-based approaches, namely TD3+BC and Onestep RL. 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 yet 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. In summary, we believe that our analysis provides improved clarity on RvS, and also presents a number of avenues for future investigation that could lead to RL methods that are both simpler and more performant.

## 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.

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.

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.

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.

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.

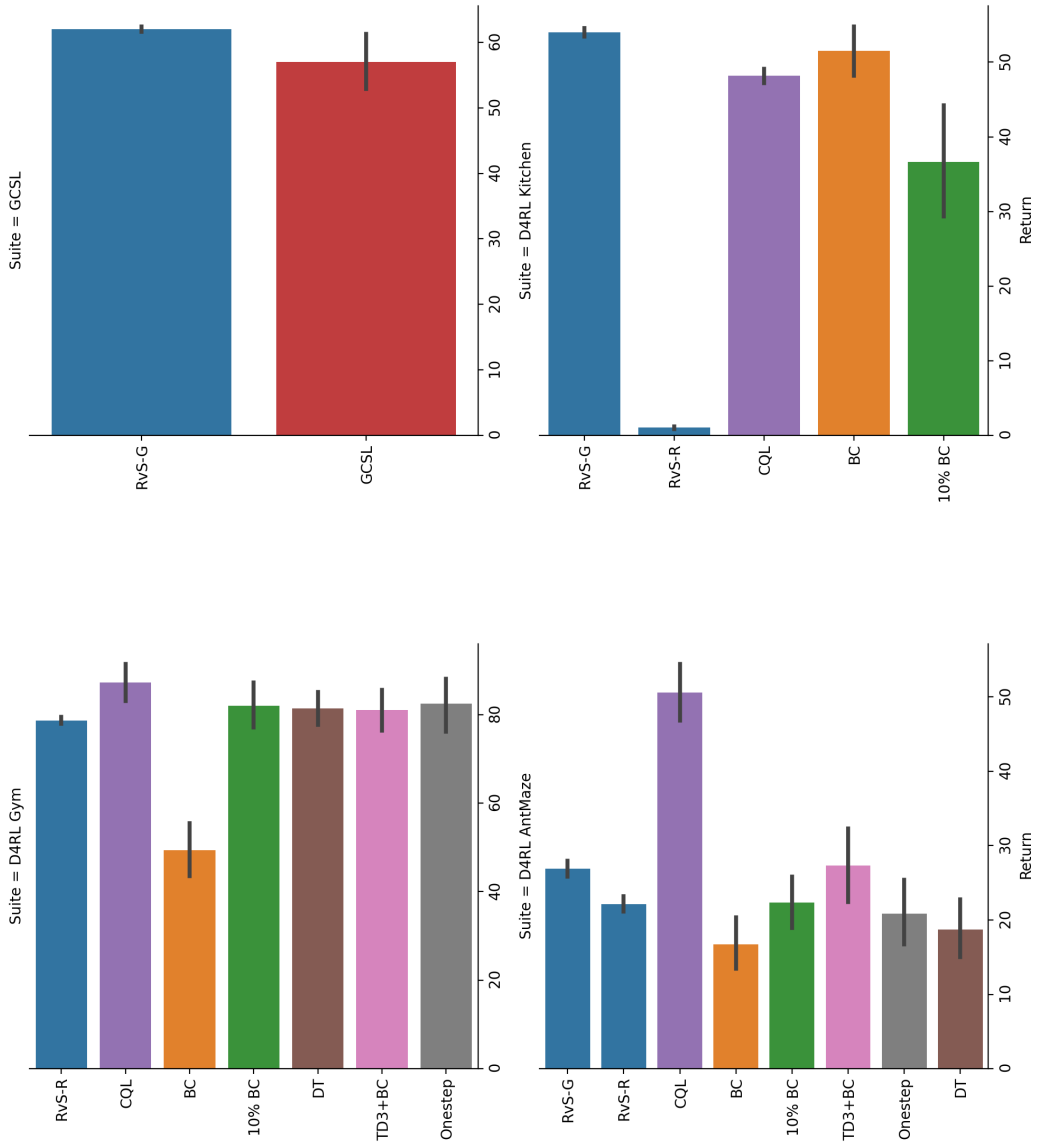
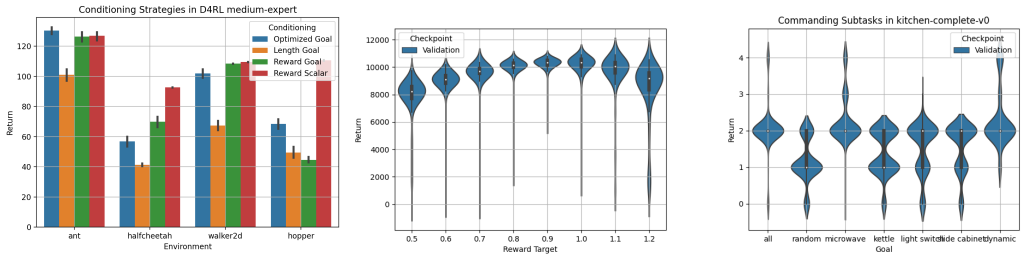



Figure 6: Same plot as Fig. 4, but enlarged to improve readability. Comparisons between goal-conditioned (RvS-G) and reward-conditioned (RvS-R) methods, as well as prior approaches and baselines. RvS results are averaged over 5 random training seeds, 200 rollouts, and multiple tasks.

## A Performance Breakdown for Each Task

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

| Suite | Algorithm<br>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-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 |
| | 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 |
| | 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 |
| 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 7: (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 7a, 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.