COG: Connecting New Skills to Past Experience with Offline Reinforcement Learning

Avi Singh, Albert Yu, Jonathan Yang, Jesse Zhang, Aviral Kumar, Sergey Levine
University of California, Berkeley
{avisingh,albertyu,jy2370,jessezhang,aviralk,svlevine}@berkeley.edu

Abstract: Reinforcement learning has been applied to a wide variety of robotics problems, but most applications involve collecting data from scratch for each new task. Since the amount of robot data we can collect for any single task is limited by time and cost considerations, the learned behavior is typically narrow: the policy can only execute the task in the handful of scenarios it was trained on. What if there was a way to incorporate a large amount of prior data, either from previously solved tasks or from unsupervised or undirected environment interaction, to extend and generalize learned behaviors? While most prior work on extending robotic skills using pre-collected data focuses on building explicit hierarchies or skill decompositions, we show in this paper that we can reuse prior data to extend new skills simply through dynamic programming. We show that even when the prior data does not actually succeed at solving the new task, it can still be utilized for learning a better policy, by providing the agent with a broader understanding of the mechanics of its environment. We demonstrate the effectiveness of our approach by chaining together several behaviors seen in prior datasets for solving a new task. Our hardest experimental setting involves composing four robotic skills in a row: picking, placing, drawer opening, and grasping, where a +1/0 sparse reward is provided only on task completion. We train our policies in an end-to-end fashion, mapping high-dimensional image observations to low-level robot control commands, and present results in both simulated and real world domains. Additional materials and source code can be found on our project website: `https://sites.google.com/view/cog-rl`.

Keywords: offline reinforcement learning, deep learning, offline datasets

1 Introduction

Consider a robot that has been trained using reinforcement learning (RL) to take an object out of an open drawer. It learns to grasp the object and pull it out of the drawer. If the robot is then placed in a scene where the drawer is instead closed, it will likely fail to take the object out, since it has not seen this scenario or initial condition before. For robotic systems to reason effectively in such scenarios, we might expect that methods based on hierarchies or explicit skill decomposition would be needed to integrate a drawer opening skill with the grasping behavior. But what if simply combining previously collected (and *unlabeled*) robot interaction data, which might include drawer opening and other behaviors, together with offline RL methods [1], can allow these behaviors to be combined *automatically*, without any explicit separation into individual skills? In this paper, we study how model-free RL algorithms can utilize prior data to extend and generalize learned behaviors, incorporating relevant segments of experience from this prior data as needed at test-time.

Standard online RL algorithms typically require a tight interaction loop between data collection and policy learning. There has been a significant amount of recent interest in devising methods for offline RL [1, 2, 3, 4, 5, 6], which leverage previously collected datasets without environment interaction. In this paper, we build on recent advances in offline RL, and show that a "data-driven" RL paradigm allows us to build real-world robotic learning systems that can learn increasingly flexible and general skills. We will show that collecting a large dataset that covers a large repertoire of skills provides a useful foundation for learning new tasks, enabling learned policies to perform multi-stage tasks

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

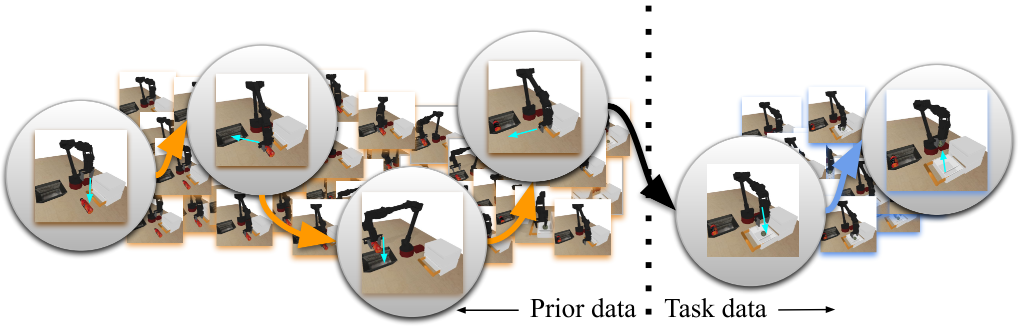

Figure 1: **Incorporating unlabeled prior data into the process of learning a new skill.** We present a system that allows us to extend and generalize robotic skills by using unlabeled prior datasets. Learning a new skill requires collecting some task-specific data (right), which may not contain all the necessary behaviors needed to set up the initial conditions for this skill in a new setting (e.g., opening a drawer before taking something out of it). The prior data (left) can be used by the robot to automatically figure out that, when it encounters a closed drawer at test time, it should open it before picking up the object. The task data does not contain drawer opening, and the prior data does not contain any examples of lifting the new object.

from previously unseen conditions, despite never having seen all of the stages together in a single episode. Consider the drawer example mentioned previously. With our method, the robot can draw on its prior experience of drawer opening to take an object out of the drawer even if the drawer is initially closed, by figuring out that opening the drawer brings it into a state from which the grasping skill can be successfully executed. Furthermore, even if there is an obstruction in front of the drawer, the robot can once again draw on the prior data, which might also include pick and place behaviors, to remove the obstruction. Crucially, this does not require the robot to practice drawer opening again when learning a new task – it can collect data for the new task in isolation, and combine it with past data to compose novel behaviors. We illustrate this idea in Figure 1.

How can this prior data from different tasks be utilized for learning a new task? If the prior data contains successful behavior for the new task, this would be easy. However, we show that even prior data that is unsuccessful at the new task can be useful for solving it, since it provides the policy with a deeper understanding of the mechanics of the world. In contrast to prior works that approach this problem from a hierarchical skill learning [7, 8] or planning perspective [9, 10], we show that this "understanding" can be acquired without any explicit predictive model or planning, simply via model-free RL. Our method builds on a recently proposed model-free offline RL method called conservative Q-learning (CQL) [3]. We extend CQL to our problem setting as following: we initialize the replay buffer with our prior dataset, and assign zero reward to every transition in this dataset, since the new task is different from the prior behaviors. We then run CQL on both this prior dataset and data collected specifically for the new task and evaluate the policy on a variety of initial conditions which were not seen in the task-specific data.

Our main contribution is demonstrating that model-free offline RL can learn to combine task data with prior data, producing previously unseen combinations of skills to meet the pre-conditions of the task of interest, and effectively generalizing to new initial conditions. We call our approach **COG**: Connecting skills via Offline RL for Generalization. We describe a robotic learning system that builds on this idea to learn a variety of manipulation behaviors, directly from images. Our system is evaluated on several manipulation tasks. We first collect prior datasets consisting of imperfect scripted behaviors, such as grasping, pulling, pushing, and placing a variety of objects. This prior data is then used to learn several downstream skills: opening and closing drawers, taking objects out of drawers, putting objects in a tray, and so on. We train neural network-based policies on raw, high-dimensional image observations, and only use sparse binary rewards as supervision. We demonstrate the effectiveness of COG in both simulated domains and on a real world low-cost robotic arm.

2 Related Work

Robotic RL. RL has been applied to a wide variety of robotic manipulation tasks, including grasping objects [11, 12], in-hand object manipulation [13, 14, 15, 16], pouring fluids [17], door opening [18, 19], and manipulating cloth [20, 21]. Most of these works use online RL methods, relying on a well-tuned interaction loop between data collection and policy training, instead of leveraging prior datasets. Our paper is more closely related to Kalashnikov et al. [11], Julian et al. [22], and Cabi et al. [23], which also use offline RL and large prior datasets. However, these prior works focus on generalizing to new objects, as well as fine-tuning to handle greater variability (e.g., more object types, changes in lighting, etc.). Our work instead focuses on changes in initial conditions that require entirely different skills than those learned as part of the current task, such as opening a drawer before grasping an object.

Data-driven robotic learning. In addition to RL-based robotics, data-driven robotics in general has become increasingly popular in recent years, and several works have investigated using large-scale datasets to tackle long-standing challenges in robotics, such as grasping novel objects. However, most prior work in this category focuses on executing the same actions on novel objects [24, 25, 26, 27, 28]. In contrast, we explicitly target problems where new behavior needs to be learned to perform the task in a new scenario, and use prior interaction datasets to achieve this ability via model-free RL. Visual foresight [29] and its followups [30, 9, 31, 32] also address temporally extended tasks with large datasets by learning video prediction models, but with significantly shorter time horizons than demonstrated in our work, due to the computational demands of long-horizon video prediction. Mandlekar et al. [33] use an alternate approach of explicitly learning hierarchical policies for control, and Mandlekar et al. [34] utilize offline imitation learning to compose different demonstration trajectories together.

Offline deep RL. While offline (or "batch") RL is a well-studied area [35, 36, 37, 1], there has been a significant amount of recent interest in offline deep RL, where deep RL agents are trained on a static, pre-collected dataset without any environment interaction [1, 3, 2, 4, 5, 38, 39, 40]. These works largely focus on developing more effective offline deep RL algorithms by correcting for distributional shift [41, 3, 4, 38, 39, 40] which renders standard off-policy RL algorithms inadmissible in purely offline settings [4, 1]. In contrast, our work does not propose a new algorithm, but rather adapts existing offline RL methods to the setting where prior data from a *different* domain must be integrated into learning a new task such that it succeeds under a variety of conditions.

3 Incorporating Prior Data into Robotic Reinforcement Learning

Our goal is to develop a system that can learn new robotic skills with reinforcement learning, while incorporating diverse previously collected data to provide greater generalization. We hypothesize that, even if a *subset of* this prior data illustrates useful behaviors that are *in support* of the new skill, including this data into the training process via offline RL can endow the robot with the ability to reason, such that when the conditions at test-time do not match those seen in the data for the specific new skill, the robot might still generalize because it has learned from other transitions to react intelligently.

We formalize our problem in the standard RL framework. The goal in RL is to optimize the infinite horizon discounted return $R_t = \sum_{t=0}^{\infty} \gamma^t r_t$ in a Markov decision process (MDP), which is defined by a tuple $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ represent state and action spaces, $T(s'|s, a)$ and $r(s, a)$ represent the dynamics and reward function, and $\gamma \in (0, 1)$ represents the discount factor. We operate in the *offline* RL setting as opposed to the standard online regime since we are interested in leveraging most out of prior datasets.

In most prior works on offline RL [4, 2, 5], the method is typically provided with data for the specific task that we wish to train a policy for, and the entire dataset is annotated with rewards that defines our objective for that task. In contrast, there are two distinct sources of data in our problem setting: task-agnostic, unlabeled prior data, which we denote as $\mathcal{D}_{\text{prior}}$, and task-specific data, which we denote as $\mathcal{D}_{\mathbb{T}}$, where $\mathbb{T}$ represents our task. The datapoints in $\mathcal{D}_{\text{prior}}$ simply consist of (s, a, s') transitions, and do not have any associated reward labels. While we cannot train a policy to achieve any particular objective from this data alone, it is informative about the dynamics of the MDP where the data was collected. On the other hand, the datapoints in $\mathcal{D}_{\mathbb{T}}$ consist of (s, a, s', r) tuples, and can be used for learning a policy that maximizes the observed reward. To summarize, the input and output of our problem setting are as follows:

Input: Datasets $\mathcal{D}_{\text{prior}}$ (with no reward annotations), $\mathcal{D}_{\mathbb{T}}$ (with sparse rewards for task $\mathbb{T}$).
Return: Policy π trained to execute task $\mathbb{T}$, which should be able to generalize broadly to new initial conditions. We would like to leverage $\mathcal{D}_{\text{prior}}$ for the latter.

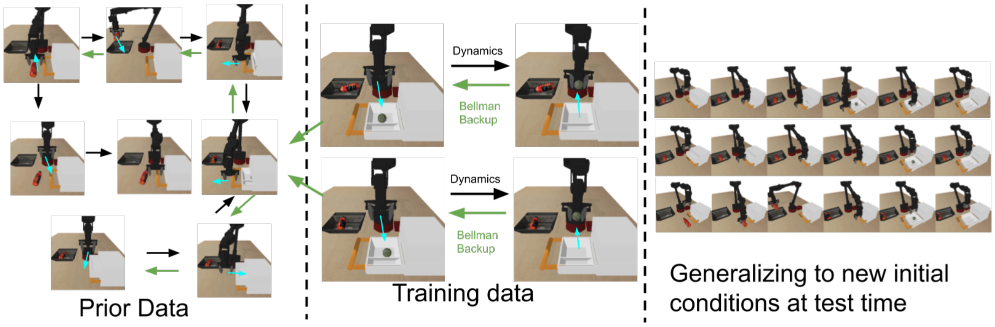

Figure 2: **Connecting new skills to past experience.** Q-learning propagates information backwards in a trajectory (middle) and by stitching together trajectories via Bellman backups from the task-agnostic prior data (left), it can learn optimal actions from initial conditions appearing in the prior data (right).

4 Connecting New Skills to Past Experience via Dynamic Programming

Our approach is conceptually very simple: use offline RL to incorporate prior data into the training for the new skill. However, the reasons why this approach should be effective in our problem setting are somewhat nuanced. In this section, we discuss how model-free dynamic programming methods based on Q-learning can be used to connect new tasks to past experience. Before presenting COG, let's first briefly revisit off-policy deep RL algorithms.

Standard off-policy deep RL methods, such as SAC [42], maintain a parametric action-value or Q-function, $Q_\theta(\mathbf{s}, \mathbf{a})$, and optionally a parametric policy, $\pi_\phi(\mathbf{a}|\mathbf{s})$, with parameters θ and ϕ, respectively. These methods typically train $Q_\theta(\mathbf{s}, \mathbf{a})$ to predict the return under π_ϕ in the policy evaluation step, and then update π_ϕ in the direction of increasing Q-values in the policy improvement step:

$$\theta^{k+1} \leftarrow \arg\min_\theta \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}_{\mathbb{T}}}\left[\Big((r(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}_{\mathbf{a}'\sim\hat{\pi}^k(\mathbf{a}'|\mathbf{s}')}[\hat{Q}^k(\mathbf{s}',\mathbf{a}')]) - Q_\theta(\mathbf{s},\mathbf{a})\Big)^2\right] \text{ (policy evaluation)}$$

$$\phi^{k+1} \leftarrow \arg\max_\phi \mathbb{E}_{\mathbf{s}\sim\mathcal{D}_{\mathbb{T}},\mathbf{a}\sim\pi^k_\phi(\mathbf{a}|\mathbf{s})}\left[\hat{Q}^{k+1}_\theta(\mathbf{s},\mathbf{a})\right] \text{ (policy improvement)}$$

To see how this model-free procedure can help us stitch together different behaviors (which we will define shortly), we start with some intuition about Q-learning. Q-learning propagates information backwards through a trajectory. State-action pairs at the end of a trajectory with a high reward are assigned higher values, and these values propagate to states at the beginning of the trajectory. Recall the example from Section 1, where the goal is to take an object out of a drawer. Assume that a reward of +1 is obtained when the object has been taken out of the drawer, and otherwise the reward is 0. Under this reward, the state where the object has been taken out will have the highest value, and the state where the robot has grasped the object (but not yet lifted it) will have a slightly lower value, since it is a few timesteps away from the successful completion of the task. The initial state, where the robot gripper is far away from the object, will have very low value. See Figure 2 (right) for an illustration of such states. However, the initial state will still have a non-zero value, since the Q-function "understands" that it is possible to reach high-reward states. Any state for which there does not exist a valid path to a high-reward state will have a value that is equal to zero, and all other states will have non-zero values, decreasing exponentially (in the discount) with distance to the successful state where the object is out of the drawer.

What happens if at test-time, the robot is asked to perform the task from some new state that was not seen in $\mathcal{D}_{\mathbb{T}}$, such as a state where the drawer is closed? Such a state would either have a value of zero or, even worse, an arbitrary value, since it is outside of the training distribution. Therefore, the robot would likely not succeed at the task from this situation. Of course, we could train the new skill from a wider range of initial states, but if each skill must be learned from every possible starting state, it will quickly become prohibitively costly to train large skill repertoires.

What if the Q-learning method was now augmented with an additional large dataset that contains a wide variety of other behaviors, but *does not contain any data for the new task*? Such a dataset can still help us learn a much more useful policy, even in the absence of reward annotation: it can provide us with trajectories that (approximately) connect states not observed in $\mathcal{D}_{\mathbb{T}}$ (e.g., a closed drawer) to states appearing in successful executions in $\mathcal{D}_{\mathbb{T}}$ (e.g., open drawer), as shown in Figure 2. If the prior dataset $\mathcal{D}_{\text{prior}}$ is large enough, it can inform the policy of different ways *of reaching* *states from which the new task is solvable*. For example, if an object blocks the drawer, and the prior dataset contains pick and place trajectories, then the policy can reason that it can unblock the drawer before opening it. Model-free Q-learning alone, without any skill decomposition or planning, can propagate values from $\mathcal{D}_{\mathbb{T}}$ into $\mathcal{D}_{\text{prior}}$, allowing us to learn a policy that can execute the task from a much broader distribution of initial states without actually seeing full executions of the task from these states. Even without a single trajectory that both opens the drawer and takes the object out, as long as there is a non-zero overlap between $\mathcal{D}_{\text{prior}}$ and $\mathcal{D}_{\mathbb{T}}$, Q-learning can still learn from $\mathcal{D}_{\text{prior}}$.

Offline RL via conservative Q-learning (CQL). In order to incorporate the prior data $\mathcal{D}_{\text{prior}}$ into the RL process, we require an algorithm that can effectively utilize such prior data without actually interacting with the environment from the same initial states. Standard off-policy Q-learning and actor-critic algorithms are susceptible to out-of-distribution actions in this setting [1, 4, 43]. We instead utilize the conservative Q-learning (CQL) [3] algorithm that additionally penalizes Q-values on out-of-distribution actions during training. CQL learns Q-functions, $Q_\theta(\mathbf{s}, \mathbf{a})$, such that the expected policy value under Q_θ lower-bounds the true policy value π_ϕ, by minimizing the log-sum-exp of the Q-values at each state $\mathbf{s}$, while maximizing the expected Q-value on the dataset action distribution, in addition to standard Bellman error training as shown in Equation 1. The training objective shown in Equation 1 is the variant of CQL used in this paper:

$$\min_{Q}\ \alpha\mathbb{E}_{\mathbf{s}\sim\mathcal{D}_{\text{prior}}\cup\mathcal{D}_{\mathbb{T}}}\left[\log\sum_{\mathbf{a}}\exp(Q(\mathbf{s},\mathbf{a})) - \mathbb{E}_{\mathbf{a}\sim\mathcal{D}_{\text{prior}}\cup\mathcal{D}_{\mathbb{T}}}\left[Q(\mathbf{s},\mathbf{a})\right]\right] + \frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}_{\text{prior}}\cup\mathcal{D}_{\mathbb{T}}}\left[\left(Q - \mathcal{B}^{\pi_k}\bar{Q}\right)^2\right]. \tag{1}$$

We instantiate CQL as an actor-critic algorithm. The policy improvement step remains unchanged from the previously discussed standard off-policy RL methods.

5 End-to-End Robotic Learning with Prior Datasets

In this section, we discuss how our method can be instantiated in a practical robotic learning system.

MDP for robotic skills. The state observation $s \in \mathcal{S}$ consists of the robot's current camera observation (an RGB image of size 48 × 48 for simulated experiments, and 64 × 64 for real robot experiments) and the current robot state. The robot state consists of the end-effector pose (cartesian coordinates and Euler angles), and the extent to which the gripper is open (represented using a continuous value). The action space $\mathcal{A}$ consists of six continuous actions and two discrete actions. The six continuous actions correspond to controlling the end-effector's 3D coordinates and its orientation. The first discrete action corresponds to opening or closing the gripper, while the second discrete action executes a return to the robot's starting configuration. We use sparse reward functions for all of our tasks: a reward of +1 is provided when a task has been executed successfully, while a zero reward is provided for all other states. We do not have any terminal states in our MDP.

Data collection. Our prior data collection takes place before any task-specific learning has happened. Since a completely random policy will seldom execute behaviors of interest, we bias our data collection policy towards executing more interesting behavior through the use of weak scripted policies: these policies typically have a success rate of 30-50% depending on the complexity of the task they are performing. More details on the scripted policies can be found in Appendix A.1.

Neural network architectures. Since we learn to stitch together behaviors directly from raw, visual observation inputs, we utilize convolutional neural networks (ConvNets) to represent our policy π_ϕ and the Q-function Q_θ. For both simulated and real world experiments, we use a 8-layer network detailed in Appendix A.2.

6 Experiments

We aim to answer the following questions through our experiments: **(1)** Can model-free RL algorithms effectively leverage prior, task-agnostic robotic datasets for learning new skills? **(2)** Can our learned policies solve new tasks, especially from novel initial conditions, by stitching together behavior observed during training? **(3)** How does our approach compare to alternative methods for incorporating prior data (such as behavior cloning)? **(4)** Is the addition of prior data essential for learning to solve the new task? To this end, we evaluate our approach on a number of long-horizon, multi-step reasoning robotic tasks with different choices of $\mathcal{D}_{\mathbb{T}}$ and $\mathcal{D}_{\text{prior}}$ and then perform an ablation study to understand the benefits of incorporating unlabeled offline datasets into robotic learning systems via offline reinforcement learning methods.

6.1 Experimental Setup

We evaluate our approach in simulation (see Figures 3 and 4) and on a real-robot task with a WidowX low-cost arm (see Figure 6).

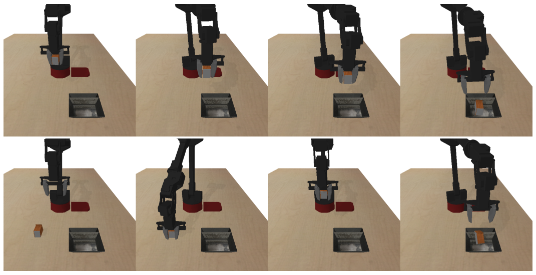

Figure 3: **Picking and placing**. Example executions from our *learned* policy. The first row shows the training condition, where the robot starts out already holding the object, and it only needs to place it in the tray. In the second condition (shown in second row), the robot must first grasp the object before placing it into the tray.

Pick and place. Our first simulated environment is shown in Figure 3. It consists of a 6-DoF WidowX robot in front of a tray containing a small object and a tray. The objective is to put the object inside the tray. The reward is +1 when the object has been placed in the box, and zero otherwise. A simple initial condition for this task involves the robot already holding the object at the start of the episode, while a harder initial condition is when the robot has to first pick up the object. For this simplified experimental setting, the prior data consists of 10K grasping attempts from a randomized scripted policy (that has a success rate of about 40%, details of this policy are in Appendix A.1). Note that we do not provide labels indicating the success of any attempt. The task-agnostic prior dataset also consists of behaviors that may be irrelevant for the task. The task-specific data consists of 5K placing attempts from a different scripted policy (with a high success rate of over 90%, since the tray position is unchanged across trials), and these trajectories are labeled with rewards. Note that there is no single trajectory in our dataset that solves the complete pick and place task, but the prior and task-specific datasets have a non-zero overlap in their state distribution.

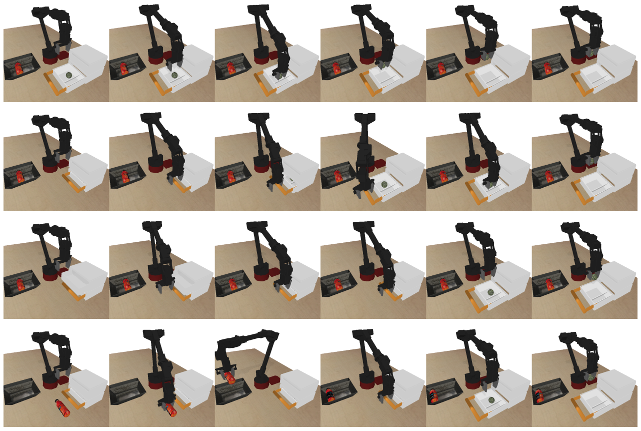

Figure 4: **Grasping from the drawer with our learned policy.** The first row shows the training condition, which requires grasping from an open drawer. The robot only needs to grasp the object and take it out of the drawer to get a reward. The subsequent rows show the harder test-time initial conditions which require, respectively: opening the drawer before taking out the object, closing the top drawer before opening the bottom one and taking out the object, and removing an obstruction (red) bottle before opening the drawer. COG learns a policy that succeeds 70-75% of the time for each type of initial condition, despite only having seen the object grasping from the open drawer.

Grasping from a drawer. Our second and more complex simulated environment is shown in Figure 4. It consists of a 6-DoF WidowX robot and a larger variety of objects. The robot can open or close a drawer, grasp objects from inside the drawer or on the table, and place them anywhere in the scene. Some of these behaviors require various preconditions. For example, grasping an object from the drawer might require opening that drawer, which in turn might require moving an obstruction out of the way. The task here consists of taking an object out of a drawer, as shown in Figure 4. A reward of +1 is obtained when the object has been taken out, and zero otherwise. When learning the new task, the drawer always starts out open. The more difficult test conditions include ones where the drawer starts out closed, the top drawer starts out open (which blocks the handle for the lower drawer), and an object starts out in front of the closed drawer, which must be moved out of the way before opening. These settings are illustrated in Figure 4. The prior data for this environment is collected from a collection of scripted randomized policies. These policies opening and closing both drawers with 40-50% success rates, grasp objects in the scene with about a 70% success rate, and place those objects at random places in the scene (with a slight bias for putting them in the tray). The prior data does *not* contain any interactions with the object inside the drawer and also contains data irrelevant to solving

Task & Initial Condition	No prior	BC			SAC	COG (ours)
	data	init	all	oracle		
place in box						
object in gripper	**1.00 (0.00)**	**1.00 (0.00)**	0.94 (0.01)	0.95 (0.01)	0.00	**1.00 (0.00)**
object in tray	0.00 (0.00)	0.00 (0.00)	0.02 (0.00)	0.02 (0.01)	–	**0.96 (0.04)**
grasp from drawer						
open drawer	0.98 (0.01)	**0.99 (0.00)**	0.63 (0.01)	0.82 (0.01)	0.00	0.98 (0.02)
closed drawer	0.00 (0.00)	0.00 (0.00)	0.23 (0.01)	0.27 (0.03)	–	**0.68 (0.07)**
blocked drawer 1	0.00 (0.00)	0.00 (0.00)	0.34 (0.03)	0.35 (0.03)	–	**0.78 (0.07)**
blocked drawer 2	0.00 (0.00)	0.00 (0.00)	0.22 (0.03)	0.25 (0.01)	–	**0.76 (0.09)**

Table 1: **Results for simulated experiments.** Mean (Standard Deviation) success rates of the learned policies for our method (COG), its ablations, and prior work. For the grasping from drawer task, blocked drawer 1 and 2 are initial conditions corresponding to the third and fourth rows of Figure 4. Note that COG successfully performs both tasks in the large majority of cases, from all initial conditions. SAC (–) diverged in our runs.

the task, such as behavior that blocks the drawer by placing objects in front of it. There are 1.5M datapoints (transitions) in the prior dataset and 300K datapoints in the task-specific dataset.

Baselines and comparisons. We compare COG to: **(1)** pre-training a policy via behavioral cloning on the prior data and then fine-tuning with offline RL on the new task, denoted as **BC-init**, **(2)** a naïve behavioral cloning baseline, denoted as **BC**, which trains with BC on all data, **(3)** an "oracle" version of behavioral cloning that is provided with handpicked successful trajectories on the new task, denoted as **BC-oracle** which approximates an upper bound on performance of selective cloning methods on a task, **(4)** a standard baseline off-policy RL method, **SAC** [42], and finally **(5)** an ablation of our method without any prior data, indicated as **no prior data**.

6.2 Empirical Results

Simulation results. The results for our simulation experiments are summarized in Table 1. For all methods, we train the policy via offline RL until the success rate converges, and evaluate the final policy on 250 trials, after training has converged. We then average these success rates across three random seeds for each experiment. Detailed learning curves can be found in Appendix C. We first note that our data-driven approach generally performs well for all initial conditions on both tasks. The policy is able to leverage the prior data to automatically determine that a closed drawer should be opened before grasping, and obstructions should be moved out of the way prior to drawer opening, despite never having seen complete episodes that involve both opening the drawer and taking out the object, and not having any reward or success labels in the prior data.

COG also significantly outperforms the behavioral cloning baselines, ablations, and a standard off-policy SAC algorithm for all novel initial conditions. Since we use an entropy-regularized stochastic policy for the behavioral cloning variants (BC in Table 1), the baseline methods are able to achieve a non-zero success rate, but the performance is far below what is achieved by COG. This is likely due to their inability to distinguish between behavior that supports the new task and behavior that is meaningful but not useful at the current time (for example, grasping an arbitrary object in the drawer task is not useful if that object is not blocking the drawer). Even when a behavioral cloning method is provided with handpicked successful trajectories for the new task (denoted "BC-oracle" in Table 1), it is unable to complete the new task. Unsurprisingly, an ablation of our method that does not make use of prior data is only able to solve the task from states that were previously seen in the task-specific dataset. BC-init, which pre-trains the policy on the prior data via cloning followed by RL on the task data, is unable to solve the task for unseen initial conditions. We believe this is due to catastrophic forgetting of behavior learned during pre-training, and this adversely affects the policy's performance when evaluated on new initial conditions.

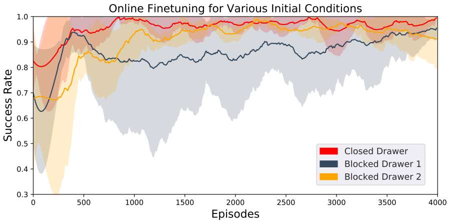

Figure 5: **Results from online fine-tuning.** We see that online fine-tuning further improves the performance of the learned policy, bringing it to over 90% success rates for all possible initial conditions for the drawer task, and only requires a small amount of additional data.

Online fine-tuning. While our method is able to obtain high success rates from offline learning alone, we also evaluate whether learned policies can be further improved via online fine-tuning. The results from these online fine-tuning experiments are shown in Figure 5. We see that for all of the novel initial conditions for the drawer task, the policy is able to achieve a success rate of over 90% from collecting only a small amount of additional episodes (500-4000, depending on the task). In our real world experimental setup (which we describe below), we are able to collect 3K episodes in a single day (autonomously), making this requirement quite feasible for real world problems. We compare this fine-tuning experiment against fine-tuning with SAC (starting from a behavior cloned policy), which performed substantially worse (see Figure 9 in Appendix B), likely because the BC policy has low initial performance and because the Q-function for SAC needs to be learned from scratch in this setting. Prior work has also observed that fine-tuning a behavior-cloned policy with SAC typically leads to some unlearning at the start, which further reduces the performance of this policy [44]. More details can be found in Appendix B.

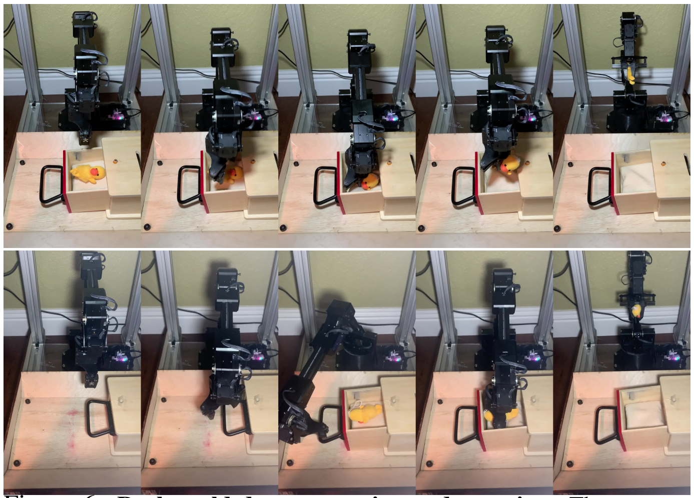

Figure 6: **Real world drawer opening and grasping.** The top row shows the training condition, which requires grasping an object from an open drawer. The bottom row shows a rollout of the learned policy in the test condition, where the drawer starts closed. Note that during training, the learned policy never saw a complete opening and grasping trajectory.

Real-world evaluation. Our real world setup (see Figure 6) consists of a WidowX robotic arm in front of a drawer, and an object inside the drawer. The task is to take the object out of the drawer. As before, the reward is +1 on completion, zero otherwise. During training for the new task, the drawer starts open. The prior dataset consists of drawer opening and closing from a scripted randomized policy (details in Appendix A.1). As before, the prior dataset has no reward labels and no instances of the new task (object grasping). The task-specific data is collected using a scripted grasping policy, which has a success rate of about 50%. The prior dataset consists of 80K transitions collected over 20 hours, and the new task dataset has 80K transitions (4K grasp attempts) collected over 34 hours. Our learned policy succeeds in **7/8** trials when the drawer starts out closed, and substantially outperforms the BC-oracle baseline, which **never** succeeds on this real-world task.

7 Discussion

We showed how model-free reinforcement learning can utilize prior data to improve the generalization of learned skills, effectively stitching together behaviors so that, when the initial conditions at test time differ from those under which a new task was learned, the agent can still perform the task by stitching together appropriate behaviors seen in the prior data. For instance, a robot that learned to grasp an object from an open drawer can automatically learn to open the drawer at test time when it starts closed, without ever having seen a complete opening and grasping trajectory. This sort of composition of skills is typically seen as the domain of model-based planning, and often approached in learning-based control from the standpoint of hierarchy. Our approach suggests that such seemingly complex multi-stage behaviors could emerge automatically as a consequence of model-free reinforcement learning with the right training data. In effect, we show that model-free RL can use prior data to learn the "mechanics" of the world, without any explicit hierarchy or model representation. Our initial instantiation of this idea does leave considerable room for future research. COG employs a very naïve approach for labeling rewards in the prior data, and more sophisticated reward inference methods could also improve performance. More broadly, we believe that further exploration into the ways that model-free RL methods can enable acquisition of implicit mechanical models of the world is a promising direction for data-driven robotics.

Acknowledgements

We thank members of the Robotics and AI Lab (RAIL) at UC Berkeley for providing feedback on early drafts of this paper. We also thank the anonymous reviewers for their constructive comments. This research was supported by the Office of Naval Research, the DARPA Assured Autonomy program, NSF IIS-1651843, and Berkeley DeepDrive, with compute support provided by Amazon, Nvidia and Google.

References

[1] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[2] S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018.

[3] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[4] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *NeurIPS*, 2019.

[5] R. Agarwal, D. Schuurmans, and M. Norouzi. An optimistic perspective on offline reinforcement learning. 2019.

[6] N. Jaques, A. Ghandeharioun, J. H. Shen, C. Ferguson, A. Lapedriza, N. Jones, S. Gu, and R. Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*, 2019.

[7] N. Fazeli, M. Oller, J. Wu, Z. Wu, J. B. Tenenbaum, and A. Rodriguez. See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion. *Science Robotics*.

[8] S. Nair and C. Finn. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. *arXiv preprint arXiv:1909.05829*, 2019.

[9] F. Ebert, C. Finn, S. Dasari, A. Xie, A. X. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *CoRR*, 2018.

[10] B. Eysenbach, R. Salakhutdinov, and S. Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *NeurIPS*, pages 15220–15231, 2019.

[11] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning (CoRL)*, 2018.

[12] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. 2018.

[13] OpenAI. Learning dexterous in-hand manipulation. In *arXiv preprint arXiv:1808.00177*, 2018.

[14] H. van Hoof, T. Hermans, G. Neumann, and J. Peters. Learning robot in-hand manipulation with tactile features. 2015.

[15] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *RSS*, 2018.

[16] V. Kumar, A. Gupta, E. Todorov, and S. Levine. Learning dexterous manipulation policies from experience and imitation. *CoRR*, abs/1611.05095, 2016.

[17] C. Schenck and D. Fox. Visual closed-loop control for pouring liquids. In *International Conference on Robotics and Automation (ICRA)*, 2017.

[18] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar, and S. Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. In *IROS*, 2017.

[19] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine. Path integral guided policy search. In *IROS*.

[20] J. Matas, S. James, and A. J. Davison. Sim-to-real reinforcement learning for deformable object manipulation. In *Conference on Robot Learning (CoRL)*, 2018.

[21] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine. End-to-end robotic reinforcement learning without reward engineering. *Robotics: Science and Systems*, 2019.

[22] R. Julian, B. Swanson, G. S. Sukhatme, S. Levine, C. Finn, and K. Hausman. Efficient adaptation for end-to-end vision-based robotic manipulation. *arXiv arXiv:2004.10190*, 2020.

[23] S. Cabi, S. G. Colmenarejo, A. Novikov, K. Konyushkova, S. E. Reed, R. Jeong, K. Zolna, Y. Aytar, D. Budden, M. Vecerík, O. Sushkov, D. Barker, J. Scholz, M. Denil, N. de Freitas, and Z. Wang. A framework for data-driven robotics. *CoRR*.

[24] D. Kappler, J. Bohg, and S. Schaal. Leveraging big data for grasp planning. In *International Conference on Robotics and Automation*. IEEE, 2015.

[25] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 3406–3413. IEEE, 2016.

[26] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *IJRR*, 2018.

[27] J. Mahler, F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kröger, J. J. Kuffner, and K. Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In D. Kragic, A. Bicchi, and A. D. Luca, editors, *ICRA*, 2016.

[28] A. Gupta, A. Murali, D. Gandhi, and L. Pinto. Robot learning in homes: Improving generalization and reducing dataset bias, 2018.

[29] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017.

[30] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv arXiv:1812.00568*, 2018.

[31] A. Xie, F. Ebert, S. Levine, and C. Finn. Improvisation through physical understanding: Using novel objects as tools with visual foresight. *arXiv preprint arXiv:1904.05538*, 2019.

[32] Y. Hristov, A. Lascarides, and S. Ramamoorthy. Interpretable latent spaces for learning from demonstration. *arXiv preprint arXiv:1807.06583*, 2018.

[33] A. Mandlekar, F. Ramos, B. Boots, L. Fei-Fei, A. Garg, and D. Fox. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. *arXiv preprint arXiv:1911.05321*, 2019.

[34] A. Mandlekar, D. Xu, R. Martín-Martín, S. Savarese, and L. Fei-Fei. Learning to generalize across long-horizon tasks from human demonstrations, 2020.

[35] S. Lange, T. Gabel, and M. A. Riedmiller. Batch reinforcement learning. In M. Wiering and M. van Otterlo, editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 45–73. Springer, 2012.

[36] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149, 2003.

[37] R. Munos. Error bounds for approximate value iteration. In *Proceedings of the National Conference on Artificial Intelligence*.

[38] Y. Wu, G. Tucker, and O. Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[39] X. B. Peng, A. Kumar, G. Zhang, and S. Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[40] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

[41] J. Fu, A. Kumar, M. Soh, and S. Levine. Diagnosing bottlenecks in deep Q-learning algorithms. *arXiv preprint arXiv:1902.10250*, 2019.

[42] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, V. K. Jie Tan, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. Technical report, 2018.

[43] A. Kumar. Data-driven deep reinforcement learning. `https://bair.berkeley.edu/blog/2019/12/05/bear/`, 2019. BAIR Blog.

[44] A. Nair, M. Dalal, A. Gupta, and S. Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Appendices

A Experimental Setup Details

Both our real and simulated environments use the following 8-dimensional control scheme:

```
[x,y,z,alpha,beta,gamma,gripperOpen,moveToNeutral]
```

where the `x,y,z` dimensions command changes in the end-effector's position in 3D space, `alpha,beta,gamma` command changes in the end-effector's orientation, `gripperOpen` is a continuous value from $[-1, 1]$ that triggers the gripper to close completely when less than -0.5 and open completely when greater than 0.5, and `moveToNeutral` is also a continuous value from $[-1, 1]$ that triggers the robot to move to its starting joint position when greater than 0.5. The code for our environments can be found on our project website: https://sites.google.com/view/cog-rl.

A.1 Data Collection Policies

We describe our scripted data collection policies in this section. More details can be found in Algorithms 1-3.

Scripted grasping. Our scripted grasping policy is supplied with the object's (approximate) coordinates. In simulation, this information is readily available, while in the real world we use background subtraction and a calibrated monocular camera to approximately localize the object. Note that this information does not need to be perfect, as we add a significant amount of noise to the scripted policy's action at each timestep. After the object has been localized, the scripted policy takes actions that move the gripper toward the object (i.e action $\leftarrow$ object_position $-$ gripper_position). Once the gripper is within some pre-specified distance of the object, it closes the gripper (which is a discrete action). Note that this distance threshold is also randomized – sampled from a Gaussian distribution with a mean of 0.04 and a standard deviation of 0.01 (in meters). It then raised the object until it is above a certain height threshold. For the simulated pick and place environment, the scripted policy for grasping obtains a success rate of 50%, while the success rate is 70% for the drawer environment. For the real world drawer environment, the scripted success rate is 30%.

Scripted pick and place. The pick part of the pick and place scripted policy is identical to the grasping policy described above. After the grasp has been attempted, the scripted policy uniformly randomly selects a point in the workspace to place the object on, and then takes actions to move the gripper above that point. Once within a pre-specified (and randomized) distance to that point, it opens the gripper. The policy is biased to sample more drop points that lie inside the tray to ensure we see enough successful pick and place attempts. Once the object has been dropped, the robot returns to its starting configuration (using the moveToNeutral action).

Scripted place. This policy is used in scenes where the robot is already holding the object at the start of the episode. The placing policy is identical to the place component of the pick and place policy described above.

Drawer opening and closing. The scripted drawer opening policy moves the gripper to grasp the drawer handle, then pulls on it to open the drawer. The drawer closing policy is similar, except it pushes on the drawer instead of pulling it. Even if the correct action for a particular task might involve only opening the drawer, we collect data (without reward labels) that involves both opening and closing the drawer during prior data collection. Gaussian noise is added to the policy actions at every timestep. After the opening/closing is completed, the robot returns to its starting configuration.

Ending scripted trajectories with return to starting configuration We ended the scripted trajectories with a return to the robot's starting configuration. We believe that this return to starting configuration increases the state-distribution overlap of the various datasets collected from scripted policies, making it possible to stitch together relevant trajectories from the prior dataset to extend the skill learned for the downstream task.

Algorithm 1 Scripted Grasping

```
threshold $\sim \mathcal{N}(0.04, 0.01)$
numTimesteps ← 30
**for** t **in** (0, numTimesteps) **do**
    objPos ← object position
    eePos ← end effector position
    objGripperDist ← distance(objPos, eePos)
    **if** objGripperDist $>$ threshold **then**
        action ← objPos $-$ eePos
    **else if** gripperOpened **then**
        action ← close gripper
    **else if** object not raised high enough **then**
        action ← lift upward
    **else**
        action ← 0
    **end if**
    noise $\sim \mathcal{N}(0, 0.2)$
    action ← action + noise
    $(s, r, s')$ ← env.step(action)
**end for**
```

Algorithm 2 Scripted Pick and Place

```
threshold, dropDistThreshold $\sim \mathcal{N}(0.04, 0.01)$
numTimesteps ← 30
**for** t **in** (0, numTimesteps) **do**
    eePos ← end effector position
    dropPos ← $\begin{cases} \text{point above box} & \text{w/ prob. } 0.5 \\ \text{point outside box} & \text{w/ prob. } 0.5 \end{cases}$
    objectDropDist ← distance(eePos, dropPos)
    **if** object not grasped **AND** objectDropDist $>$ dropDistThreshold **then**
        Execute grasp using Algorithm 1
    **else if** objectDropDist $>$ boxDistThreshold **then**
        action ← dropPos $-$ eePos
        action ← lift upward
    **else if** object not dropped **then**
        action ← open gripper
    **else**
        action ← 0
    **end if**
    noise $\sim \mathcal{N}(0, 0.2)$
    action ← action + noise
    $(s, r, s')$ ← env.step(action)
**end for**
```

Algorithm 3 Scripted Drawer Opening/Closing

```
threshold $\sim \mathcal{N}(0.04, 0.01)$
numTimesteps ← 30
**for** t **in** (0, numTimesteps) **do**
    handlePos ← handle center position
    eePos ← end effector position
    targetGripperDist ← dist(targetPos, eePos)
    **if** targetGripperDist $>$ threshold **AND** not drawerOpened **then**
        action ← targetPos $-$ eePos
    **else if** not drawerOpened (or closed) **then**
        action ← move left to open drawer, or right to close drawer
    **else if** gripper not above drawer **then**
        action ← lift upward
    **else**
        action ← moveToNeutral
        End scripted trajectory
    **end if**
    noise $\sim \mathcal{N}(0, 0.2)$
    action ← action + noise
    $(s, r, s')$ ← env.step(action)
**end for**
```

A.2 Neural Network Architectures

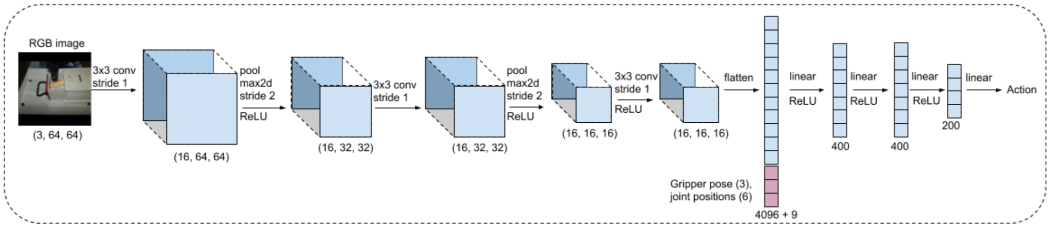

Figure 7: **Neural network architecture for real robot experiments.** Here we show the architecture for the policy network for real robot experiments. The Q-function architecture is identical, except it also has the action as an input that is passed in after the flattening step. We map high dimensional image observations to low level robot commands, i.e. desired position of the end-effector, and gripper opening/closing.

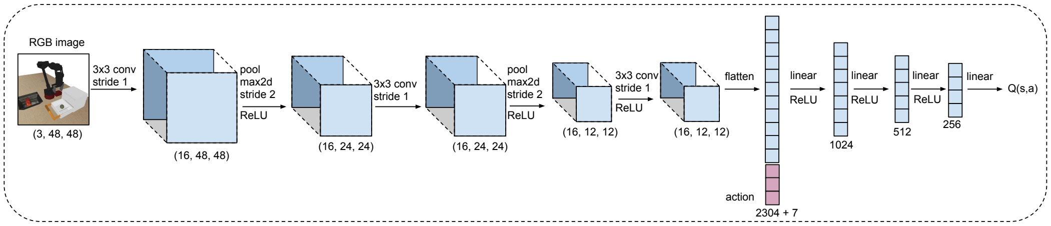

Figure 8: **Neural network architecture for simulated experiments**. Here we show the architecture for the Q-function in our simulated experiments. The policy architecture is identical, except no action is passed to the network. Note that we omit the information about the gripper position and orientation, since including this information did not seem to make a difference in our simulated experiments.

Figures 7 and 8 show the neural network architectures used in our real world and simulated experiments, respectively. We experimented with several different architectures (varying the number of convolutional layers from 2 to 4, and varying the number of filters in each layer from 4 to 16), and found these two architectures to perform the best.

A.3 Hyperparameters for Reinforcement Learning

We used the conservative Q-learning (CQL) [3] algorithm in our method. Source code can be found on our project website: `https://sites.google.com/view/cog-rl` We now present the hyperparameters used by our method below:

- **Discount factor**: 0.99 (identical to SAC, CQL),
- **Learning rates**: Q-function: 3e-4, Policy: 3e-5 (identical to CQL),
- **Batch size**: 256 (identical to SAC, CQL),
- **Target network update rate**: 0.005 (identical to SAC, CQL),
- **Ratio of policy to Q-function updates**: 1:1 (identical to SAC, CQL),
- **Number of Q-functions**: 2 Q-functions, $\min(Q_1, Q_2)$ used for Q-function backup and policy update (identical to SAC, CQL),
- **Automatic entropy tuning**: True, with target entropy set to $-\log|\mathcal{A}|$ (identical to SAC, CQL),
- **CQL version**: CQL($\mathcal{H}$) (note that this doesn't contain an additional $-\alpha \log \pi(\mathbf{a}|\mathbf{s})$ term in the Q-function backup),
- α **in CQL**: 1.0 (we used the non-Lagrange version of CQL($\mathcal{H}$)),
- **Number of negative samples used for estimating logsumexp:** 1 (instead of the default of 10 used in CQL; reduces training wall-clock time substantially when learning from image observations)

- **Initial BC warmstart period**: 10K gradient steps
- **Evaluation maximum trajectory length**: 80 timesteps for simulated drawer environment, 40 timesteps for simulated pick and place. For real world drawer environment, this value is equal to 35 timesteps.

B Comparison to BC + SAC for online fine-tuning

We compared our CQL fine-tuning results to fine-tuning a behavior-cloned policy with SAC, and observed that fine-tuning with CQL yields substantially better results. The comparison between between CQL fine-tuning, and this BC+SAC baselines is shown in Figure 9 for the grasping from a drawer task (see Figure 4), for the initial condition where the drawer starts out closed. We see that the initial SAC performance is low, which is partly due to the low success rate of the BC policy, and also because SAC typically undergoes some unlearning at the start of the fine-tuning process. This unlearning when fine-tuning with SAC has been observed in prior work [44], and is due to the fact that a randomly initialized critic is used to update the policy. For harder (i.e. long-horizon) tasks with more complicated initial conditions (such as blocked drawer 1 and blocked drawer 2), we were unable to get SAC to perform well from a BC initialization, even after we collecting over 5K new episodes.

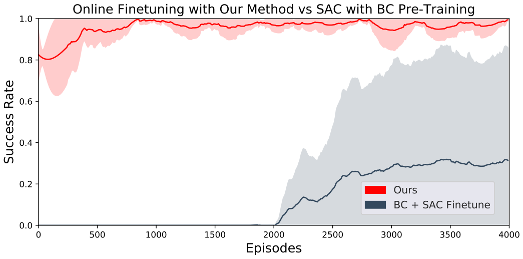

Figure 9: **Fine-tuning with CQL vs BC+SAC** We compared fine-tuning with CQL to fine-tuning a BC policy with SAC. SAC experiences some unlearning at the start (resulting in a success rate of zero at the start of training), and needs to collect a somewhat large number of samples before it can recover. Further, the variance across three random seeds was quite high for BC+SAC.

C Learning Curves

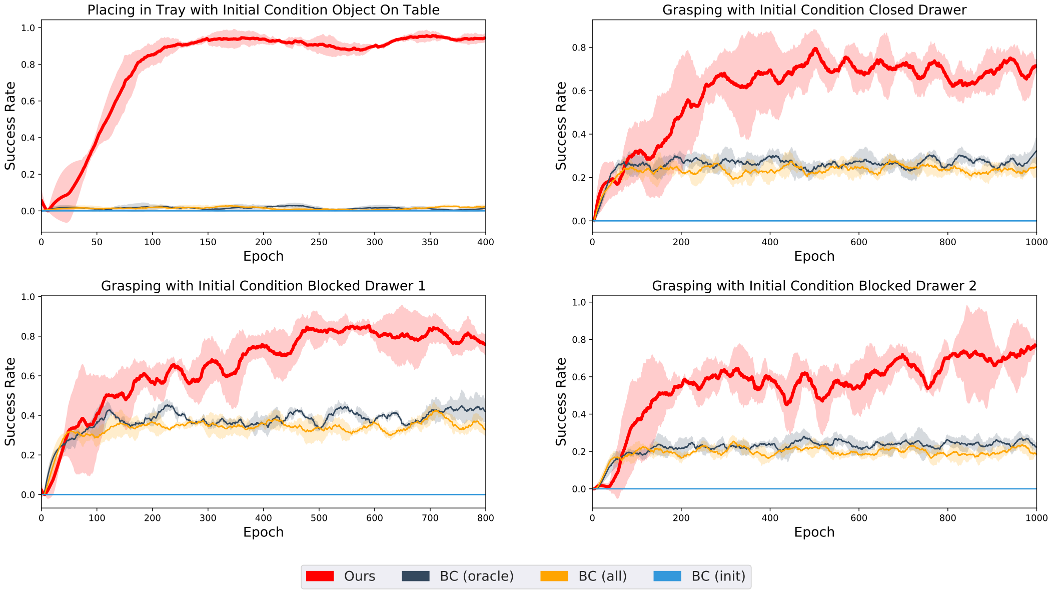

Figure 10: **Learning curves for simulated experiments by method and initial condition**. Here we compare the success rate curves of our method (COG) to the three behavioral cloning baselines in the four settings of Table 1 where prior data is essential for solving the task: the place in tray task with the object starting in the tray (upper left), as well as the grasp from drawer task with a closed drawer (upper right), blocked drawer 1 (lower left), and blocked drawer 2 (lower right).

Here are detailed learning curves for the experiments we summarized in Table 1. Note that the x-axis here denotes number of update steps made to the policy and Q-function, and not the amount of data available to the method. Since we operate in an offline reinforcement learning setting, all data is available to the methods at the start of training. We see that COG is able to achieve a high performance across all initial conditions for both the tasks. We substantially outperform comparisons to prior approaches that are based on pretraining using behavior cloning, including an oracle version that only uses trajectories with a high reward.