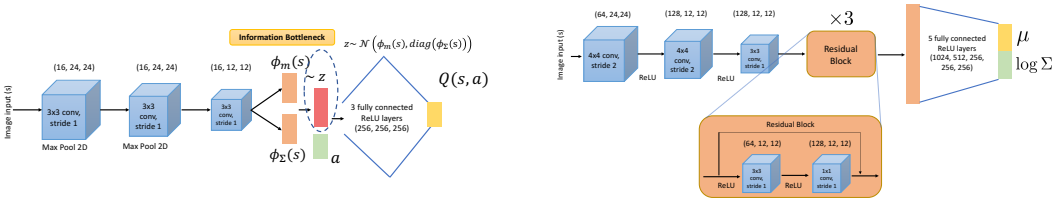


Figure 10: Architectures for addressing *(Left)* underfitting, using an information bottleneck regularizer, and *(Right)* overfitting, using a residual network.

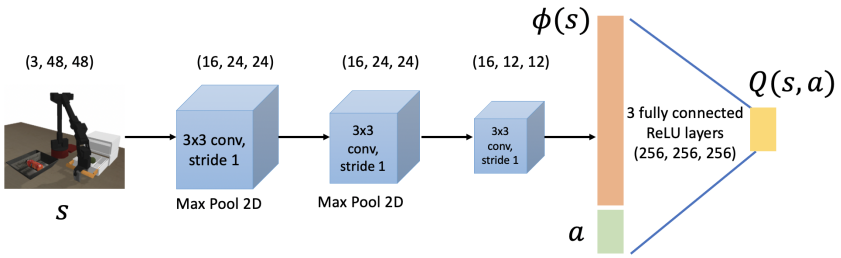


Figure 11: Standard architecture for the Q-function used in our experiments. We build on the code provided in Singh et al. [3] and utilize their default architecture.

Appendices

A Additional Discussion of Overfitting and Underfitting

In this section, we shall discuss additional details pertaining to various metrics and protocols for detecting overfitting and underfitting discussed in Section 3. We first formalize our insight as to why decreasing Q-values as a result of more gradient steps are indicative of overfitting in offline RL and then provide additional discussion about underfitting.

A.1 Overfitting in CQL

Our proposed workflow in Section 3 characterizes overfitting in CQL as a non-monotonic, first increasing and then decreasing trend in the average dataset Q-value. To understand why this trend can be used to characterize overfitting, i.e., a reduction in the test objective $J(\pi)$ (the actual return of the learned policy) as per our definition in Section 2, Table 1, we first characterize conditions under which CQL (Equation 1) Q-values averaged over the samples in the training dataset cannot exhibit a decreasing trend with more iterations of of training. To derive these conditions, we operate in the regime where the policy π_ϕ is trained to exactly maximize the Q-value, $\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi_\phi(\cdot|\mathbf{s})}[Q_\theta(\mathbf{s},\mathbf{a})]$.

Notation and Assumptions. In order to understand the trend in the average dataset Q-value observed in our experiments, we consider a slightly modified variant of Equation 1 marked with indices that denote the iteration of learning $k = 1, 2, \cdots$:

$$Q^{k+1} \leftarrow \arg\min_Q \alpha \left(\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D},\mu}[Q(\mathbf{s},\mathbf{a})] - \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q(\mathbf{s},\mathbf{a})]\right) + \frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[\left(Q - \mathcal{B}^{\pi^k}Q^k\right)^2\right], \quad (5)$$

where π^k is the policy that maximizes the current Q-function, Q^k. Thus, variant of CQL shown in Equation 5 implements *exact* policy optimization at each step of training k: $\forall \mathbf{s}, \mathbf{a}, \quad \pi^k(\mathbf{a}|\mathbf{s}) = \delta[\mathbf{a} = \arg\max_{\mathbf{a}'} Q^k(\mathbf{s},\mathbf{a}')]$. Arguably, this is closer to how CQL (and other actor-critic algorithms) are performed in practice – rather than performing a complete evaluation of the learned policy and only then performing policy improvement, these practical approaches perform alternating iterations of policy evaluation and improvement. The Q-learning variant of CQL [2] actually performs exact policy improvement for each step, which is exactly what is shown in Equation 5. As a result, we

analyze Equation 5. Our goal will be to understand the conditions under which the learned Q-values, averaged over the dataset, can exhibit a decreasing trend with more training.

Theorem A.1 (Characterizing a decreasing trend in Q-values)**.** *When running CQL using updates in Equation 5 in a tabular setting, using the policy $\pi^k(\mathbf{a}|\mathbf{s}) = \delta[\mathbf{a} = \arg\max_{\mathbf{a}'} Q^k(\mathbf{s}, \mathbf{a}')]$, the expected Q-value on the dataset, i.e., $f(k) := \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q^k(\mathbf{s}, \mathbf{a})]$, is a non-decreasing function of iteration k, i.e., $f(k+1) \geq f(k)$, whenever either of the two conditions hold:*

1. *The learned average dataset Q-value is smaller than the Monte-Carlo return of the dataset: $f(k) \leq \frac{1}{1-\gamma}\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[r(s,a)]$ (expected dataset return), or,*
2. *The gap between the maximal value of the learned Q-function $\max_{\mathbf{a}} Q^k(\mathbf{s}, \mathbf{a})$ and the Q-function value at a different action $\mathbf{a}'$ at a given state $\mathbf{s}$ in expectation over all dataset states is large enough, i.e., $\mathbb{E}_{\mathbf{s}\sim\mathcal{D}}[\max_{\mathbf{a}} Q^k(\mathbf{s}, \mathbf{a}) - Q^k(\mathbf{s}, \mathbf{a}')] \geq \zeta$, where ζ depends inversely on the density of the action $\arg\max_{\mathbf{a}} Q^k(\mathbf{s}, \mathbf{a})$ under the behavior policy $\pi_\beta(\cdot|\mathbf{s})$.*

Proof. To prove this result, we build on the analysis in Kumar et al. [2] and find that the Q-function at iteration $k+1$ can be written as follows:

$$Q^{k+1}(\mathbf{s}, \mathbf{a}) := \left(\mathcal{B}^{\pi_k} Q^k\right)(\mathbf{s}, \mathbf{a}) - \alpha\left(\frac{\mu(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right),$$

where $\pi_\beta(\mathbf{a}|\mathbf{s})$ denotes the behavior policy action-conditioned on state marginal in the dataset $\mathcal{D}$. The average Q-value in the dataset is thus given by:

$$\begin{aligned}
f(k+1) &= \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q^{k+1}(\mathbf{s}, \mathbf{a})] \\
&= \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}\left[\left(\mathcal{B}^{\pi^k} Q^k\right)(\mathbf{s}, \mathbf{a})\right] - \alpha\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi_\beta(\mathbf{a}|\mathbf{s})}\left[\frac{\mu(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right] \\
&= \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}\left[r(\mathbf{s}, \mathbf{a}) + \gamma\mathbb{E}_{\mathbf{s}'\sim P(\mathbf{s}'|\mathbf{s},\mathbf{a})}[\max_{\mathbf{a}'} Q^k(\mathbf{s}', \mathbf{a}')]\right] - 0.
\end{aligned}$$

Now, we consider the behavior of the above quantity $f(k)$, when the reward function $r(\mathbf{s}, \mathbf{a}) \geq 0$, and in particular, for our domains of interest, $\forall \mathbf{s}, \mathbf{a},\ r(\mathbf{s}, \mathbf{a}) = 0$ or $r(\mathbf{s}, \mathbf{a}) = 1$. When the initial value function $\forall \mathbf{s}, \mathbf{a},\ Q^0(\mathbf{s}, \mathbf{a}) = 0$, we now wish to characterize conditions under which the Q-function iterates $Q^1, \cdots, Q^k, \cdots$ are monotonically increasing in expectation, i.e.,

$$\forall \mathbf{s}, \mathbf{a},\ \ \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q^1(\mathbf{s}, \mathbf{a})] \leq \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q^2(\mathbf{s}, \mathbf{a})] \leq \cdots \leq \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q^k(\mathbf{s}, \mathbf{a})] \leq \cdots.$$

We can analyze this progression using mathematical induction. To first prove the base case, note that since $Q^0(\mathbf{s}, \mathbf{a}) = 0$ (initialization), $f(1) = \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[r(\mathbf{s}, \mathbf{a})]$, and

$$f(2) = \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q^1(\mathbf{s}', \mathbf{a}')] \geq \underbrace{\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[r(\mathbf{s}, \mathbf{a})]}_{=f(1)} + \gamma \underbrace{\mathbb{E}_{\mathbf{s}',\mathbf{a}'\sim\mathcal{D}P^{\pi_\beta}}[Q^1(\mathbf{s}', \mathbf{a}')]}_{\geq 0},$$

since in expectation, $\mathbb{E}_{\mathbf{s}',\mathbf{a}'\sim\mathcal{D}P^{\pi_\beta}}[Q^1(\mathbf{s}', \mathbf{a}')] = \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[r(\mathbf{s}, \mathbf{a})]$, where $(\mathbf{s}', \mathbf{a}') \sim \mathcal{D}P^{\pi_\beta} = \mathcal{D}$, since the dataset distribution is the stationary state-action visitation distribution of the behavior policy π_β. Thus, we find that $f(2) \geq f(1)$, proving the base case for induction,

Now we assume that upto a given k, $\forall j \in [k], f(j) \geq f(j-1)$. Then, our aim is to derive the condition that $f(k+1) \geq f(k)$. To show this, we write out the expressions:

$$\begin{aligned}
f(k+1) &= \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}\left[r(\mathbf{s}, \mathbf{a}) + \gamma\mathbb{E}_{\mathbf{s}'}\left[\max_{\mathbf{a}'} Q^k(\mathbf{s}', \mathbf{a}')\right]\right] \\
f(k) &= \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}\left[r(\mathbf{s}, \mathbf{a}) + \gamma\mathbb{E}_{\mathbf{s}'}\left[\max_{\mathbf{a}'} Q^{k-1}(\mathbf{s}', \mathbf{a}')\right]\right],
\end{aligned} \tag{6}$$

and then expand $f(k+1) - f(k)$:

$$\begin{aligned}
f(k+1) - f(k) &= \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[r(\mathbf{s}, \mathbf{a})] + \gamma\mathbb{E}_{\mathbf{s}'}[\max_{\mathbf{a}'} Q^k(\mathbf{s}, \mathbf{a})] - \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q^k(\mathbf{s}, \mathbf{a})] \\
&= \underbrace{\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[r(\mathbf{s}, \mathbf{a})] - (1-\gamma)f(k)}_{\text{(a)}} + \gamma\underbrace{\mathbb{E}_{\mathbf{s}\sim\mathcal{D}}\left[\max_{\mathbf{a}} Q^k(\mathbf{s}, \mathbf{a}) - \mathbb{E}_{\pi_\beta}[Q^k(\mathbf{s}, \mathbf{a})]\right]}_{\text{(b)}}.
\end{aligned}$$

First, by definition note that $(b) \geq 0$. And term $(a) \geq 0$ for iterations k where $f(k) \leq \frac{\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[r(\mathbf{s},\mathbf{a})]}{1-\gamma}$, which occurs whenever the average dataset Q-value, $f(k)$ is smaller than the dataset discounted cumulative reward. Thus, whenever the dataset Q-value is smaller than the average dataset discounted cumulative reward, $(a) \geq 0$, $(b) \geq 0$ implying that $f(k+1) \geq f(k)$.

Now, let's consider the case when the average dataset Q-value is smaller than the expected cumulative reward in the dataset and characterize the conditions under which the Q-values will exhibit a non-decreasing trend in this case. To characterize this condition, we begin with a direct difference of the expressions for $f(k)$ and $f(k-1)$ in Equation 6 and analyze the difference in Q-values from consecutive Q-function iterates at arg-max actions $\mathbf{a}'$ at the next state $\mathbf{s}'$. For all iterations k, where $(a) \leq 0$, i.e., the average dataset Q-value is higher than the dataset discounted cumulative reward, consider the expressions for $f(k+1)$ and $f(k)$ from Equation 6 again, and note that there are two cases for each state $\mathbf{s}'$ appearing in the RHS of the expressions:

Case 1: $\arg\max_{\mathbf{a}'} Q^k(\mathbf{s}',\mathbf{a}') = \arg\max_{\mathbf{a}'} Q^{k-1}(\mathbf{s}',\mathbf{a}')$**:** In this case, using the expression for the Q-function obtained in CQL, we can express Q^k as:

$$Q^k(\mathbf{s}',\mathbf{a}') - Q^{k-1}(\mathbf{s}',\mathbf{a}') = \gamma\mathbb{E}_{\mathbf{s}''}\left[\max_{\mathbf{a}''} Q^{k-1}(\mathbf{s}'',\mathbf{a}'') - \max_{\mathbf{a}''} Q^{k-2}(\mathbf{s}'',\mathbf{a}'')\right],$$

which is a similar expression to what already exists in an expansion of $f(k) - f(k-1)$ analogous to Equation 6.

Case 2: $\arg\max_{\mathbf{a}'} Q^k(\mathbf{s}',\mathbf{a}') \neq \arg\max_{\mathbf{a}'} Q^{k-1}(\mathbf{s}',\mathbf{a}')$**:** Let $\mathbf{a}_1 = \arg\max_{\mathbf{a}'} Q^k(\mathbf{s}',\mathbf{a}')$ and let $\mathbf{a}_2 = \arg\max_{\mathbf{a}'} Q^{k-1}(\mathbf{s}',\mathbf{a}')$. Then, we can split their difference as:

$$Q^k(\mathbf{s}',\mathbf{a}_1) - Q^{k-1}(\mathbf{s}',\mathbf{a}_2) = \underbrace{Q^k(\mathbf{s}',\mathbf{a}_2) - Q^{k-1}(\mathbf{s}',\mathbf{a}_2)}_{(i)} + \underbrace{Q^k(\mathbf{s}',\mathbf{a}_1) - Q^k(\mathbf{s}',\mathbf{a}_2)}_{(ii)}.$$

Term (ii) in the above expression is non-negative, since $\mathbf{a}_1$ is the action with the highest Q-value Q^k at state $\mathbf{s}'$. Term (i) in the above expression can be split further:

$$Q^k(\mathbf{s}',\mathbf{a}_2) - Q^{k-1}(\mathbf{s}',\mathbf{a}_2) := -\frac{\alpha}{\pi_\beta(\mathbf{a}_2|\mathbf{s}')} + \gamma\mathbb{E}_{\mathbf{s}''}\left[\max_{\mathbf{a}''} Q^{k-1}(\mathbf{s}'',\mathbf{a}'') - \max_{\mathbf{a}''} Q^{k-2}(\mathbf{s}'',\mathbf{a}'')\right].$$

The second term in the above expression is similar to the term in $f(k) - f(k-1)$, and thus if the offset $-\frac{\alpha}{\pi_\beta(\mathbf{a}_2|\mathbf{s}')}$ does not fully compensate for the increase due to term (ii), by induction we can claim that $f(k+1) \geq f(k)$ if the inequality holds for all $j \leq k$.

To summarize, we can group the two cases to list down conditions under which the learned average dataset Q-value *can* decrease in a given iteration k of CQL. This means that it is not necessary that the Q-values would decrease when these conditions are met, but if these conditions are not met, then the Q-values cannot necessarily decrease with more training. For a given iteration k of CQL, the average Q-value under the dataset can decrease when:

$$f(k) \geq \frac{\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[r(\mathbf{s},\mathbf{a})]}{1-\gamma} \quad \text{and} \quad \mathbb{E}_{\mathbf{s}'\sim\mathcal{D}}\left[\max_{\mathbf{a}} Q^k(\mathbf{s}',\mathbf{a}) - Q^k(\mathbf{s}',\mathbf{a}_2)\right] \leq \underbrace{\mathbb{E}_{\mathbf{s}'}\left[\frac{\alpha}{\pi_\beta(\mathbf{a}_2|\mathbf{s}')}\right]}_{\zeta}. \quad (7)$$

Thus, if the difference of Q-values across different actions in expectation over all states in the dataset is large enough, the condition in Equation 7 is not met and we would expect Q-values to increase, and not decrease. Similarly, in the phase of learning where the Q-value is smaller than the average dataset return, we would expect the Q-values to continue increasing. Thus, the average dataset Q-value should be non-decreasing if either of the two conditions in Equation 7 are not satisfied, which corresponds to conditions (1) and (2) in the theorem statement. □

Interpretation of Theorem A.1: Early stopping and the peak in Q-values. Now we shall deduce the conclusion of overfitting from Theorem A.1. The Q-values decrease only if the gap between Q-values at actions taken by two consecutive policy iterates is smaller than a quantity ζ that depends inversely on the likelihood of the action under the behavior policy. This means that if and once

the Q-function finds a good policy π, better than the behavior policy π_β, the average dataset Q-values can start to decrease if π is not close enough to π_β, since the actions from the learned policy $\pi(\mathbf{a}'|\mathbf{s}')$ will not have a high likelihood under the behavior policy $\pi_\beta(\cdot|\mathbf{s}')$, and thus the ζ term in Equation 7 can easily become larger than the gap between Q-values. Thus, we would expect that the peak in the Q-values would correspond to this a performing policy π, that is potentially different from the behavior policy. One would also expect that a decrease in the Q-function would cause the learned policy π to gradually move towards the behavior policy as this would increase $\pi_\beta(\mathbf{a}_2|\mathbf{s}')$ by selecting action $\mathbf{a}_2$ highly likely under the behavior policy and would thus reduce ζ. On the other hand, if the Q-values continuously increase, the learned Q-values are either smaller than the dataset Monte-Carlo return or exhibit high gaps between Q-values. In such scenarios, we would expect more gradient steps of policy evaluation and improvement to actually improve the policy, and more training would lead to improved performance. Thus, this discussion implies that a non-monotonic trend in Q-values is indicative of overfitting towards the behavior policy (Metric 3.1) and that policy selection can be performed near the peak of the Q-values (Guideline 3.1).

A.2 Underfitting in CQL

The metric used to characterize underfitting in Section 3 is to compute the value of TD error, $\mathcal{L}_{\text{TD}}(\theta)$ and the CQL regularizer, $\mathcal{R}(\theta)$ and inspect if these values are large either relative to a model with an increased capacity or on an absolute scale. To understand why this corresponds to underfitting, note that a large value of TD error corresponds to a Q-function that does not respect Bellman consistency conditions and hence may be arbitrarily worse, whereas a large positive value of the CQL regularizer corresponds to a Q-function that is not close to the behavior policy and hence may be choosing out-of-distribution actions. In either case, we would aim to learn a Q-function that minimizes both the TD-error and the CQL regularizer.

Minimizing only one of the two objectives is not sufficient in this setting: **(1)** a Q-function that minimizes training TD error to a small enough value but attains a large value of the CQL regularizer is not sufficient since this Q-function may take erroneously high values on out-of-distribution actions, leading to a worse policy, and, **(2)** a Q-function that minimizes the CQL regularizer to a small value and attains a high value of the training TD error may not correspond to a valid Q-function which may lead to a worse policy, potentially close to the behavior policy. As a result, our Metric 3.2 suggests tracking both of these values independently and utilizing a correction for underfitting if either of the two objectives (TD error and CQL regularizer) are not minimized to low-enough values.

Utilizing a fix for underfitting by default in CQL. Similar to supervised learning, precisely quantifying the amount of underfitting is hard in offline RL as well. It is an additional challenge in offline RL that the two objectives (TD error and CQL regularizer) may impose conflicting gradients, making it hard to identify the optimal value of these loss values. As a result, we would suggest that some of the proposed solutions for underfitting discussed in Section 4 such as utilizing more expressive architectures be used even in cases where it is ambiguous as to whether the loss values are large or not, provided that there are no clear signs of overfitting (per Metric 3.1).

B Additional Background

In this section, we provide additional background for the conservative Q-learning (CQL) [2] algorithm that we use as the base algorithm for devising our workflow. We utilize the actor-critic instantiation of CQL that trains a conservative Q-function $Q_\theta(\mathbf{s}, \mathbf{a})$ and a policy $\pi_\phi(\mathbf{a}|\mathbf{s})$ that maximizes the Q-function. This algorithm proceeds in alternating steps of policy evaluation and policy improvement and our practical instantiation of this algorithm operates as per the following (policy evaluation and policy improvement) updates:

$$\theta^{k+1} \leftarrow \arg\min_\theta \; \alpha \mathbb{E}_{\mathbf{s}\sim\mathcal{D}} \left[\log \sum_{\mathbf{a}} \exp(Q_\theta(\mathbf{s}, \mathbf{a})) - \mathbb{E}_{\mathbf{a}\sim\mathcal{D}} \left[Q_\theta(\mathbf{s}, \mathbf{a}) \right] \right] + \frac{1}{2} \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}} \left[\left(Q - \mathcal{B}^{\pi_k} \bar{Q} \right)^2 \right]$$

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

In practice, these updates are performed via alternating gradient descent on the actor ($\pi_\phi(\mathbf{a}|\mathbf{s})$) and the critic ($Q_\theta(\mathbf{s}, \mathbf{a})$). While the hyperparameter α in the update above also needs to be chosen offline, we utilize the value of $\alpha = 1.0$ from prior work, fixed across all our experiments in both simulated

and real-world domains, and focus on tuning other decisions such as network size, regularization and policy selection.

C Related Work

Robotic RL with offline datasets. Learning-based methods have been applied to a number of robotics problems, such as grasping objects [31, 32], in-hand object manipulation [33, 34, 35, 36], pouring fluids [37], door opening [38], and manipulating cloth [39]. While the majority of these works use standard online RL, a number of works have leveraged robotic datasets to train skills in addition to active environment rollouts. Kalashnikov et al. [31], Julian et al. [40], and Cabi et al. [41] use offline pre-training followed by a finetuning phase. Visual foresight [42, 43, 44, 45, 46] train a video-predictive dynamics model for offline planning. Young et al. [47], Johns [48] lean skills in an offline manner and use it for imitation. Mandlekar et al. [49, 50] learn hierarchical skills for imitation. Our work is complementary to these prior works, in that our workflow can be applied to any robotic offline RL system and we do not propose a new imitation algorithm.

Offline deep RL algorithms and model selection in offline RL. Algorithms for offline deep RL [51, 15] can be divided into three categories: those that constrain the policy to the dataset [52, 53, 16, 54, 55, 56], those that perform critic regularization [2, 20, 57] and those that train dynamics models [58, 59]. These methods have been applied in robotics, for example when learning from unlabeled data [3], robotic manipulation [60, 31], goal-conditioned RL [4, 30] and multi-task RL [1]. Rather than developing a new offline RL method, our work develops criteria and workflow rules that simplify the application of such methods to new robotics tasks. Prior work that attempts to tackle model-selection in offline RL has focused exclusively on devising off-policy evaluation (OPE) methods. These methods utilize importance sampling [61, 62, 12] or learn a dynamics model or a value function [13, 9, 7, 63] to estimate the policy return. However, empirical studies by Fu et al. [14] and Qin et al. [64] show that none of these OPE methods actually perform reliably and consistently across tasks and offline datasets of the kind we are likely to find in the real-world, and present tuning challenges of their own. Our workflow does not perform direct off-policy evaluation, and instead utilizes comparative metrics across checkpoints and training runs based on observations about the behavior of specific types of offline RL algorithms.

D Additional Experimental Details

D.1 Simulated Domains

In this section, we provide a detailed discussion of the domains used in our simulated experiments in Section 5.

Pick and place task. As detailed in Section 5, our first simulated domain 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 tray, and zero otherwise. The offline dataset consists of trajectories that grasp the object with a 35% success rate and place it with a success rate of 40%. We collected the dataset using scripted policies that we briefly discuss below. For more detail, please refer to Appendix A.1 in Singh et al. [3].

Scripted grasping policy. Our scripted policy is identical to the policy in Singh et al. [3]. This policy is supplied with the object's (approximate) coordinates and can localize the object using background subtraction. Once the policy localizes the objects, it goes to the objects by executing actions with added noise and then closes the gripper when it is within some pre-specified distance of the object. This distance threshold is randomized similar to Singh et al. [3] and the grasp can fail or succeed with about a 35% chances of success.

Scripted pick and place policy. As previously used in Singh et al. [3], our scripted pick and place policy attempts a grasp as described above, and then tries to place the object randomly at some location in the workspace. Only if it places the object on the tray does it get a +1 reward, and after placing the object, it moves up and tries to hover around by executing small magnitude random actions until the episode terminates.

Grasping from a blocked drawer. The scripted policies we use for this task are borrowed from Singh et al. [3]. These policies can open and close both the drawers with 40-50% success rates,

can grasp objects with about a 70% success rate, and place those objects at random locations in the workspace. Since we use the datasets from Singh et al. [3] directly, the prior data does *not* contain any interactions with the object inside the drawer and contains data such as behavior that blocks the drawer by placing objects in front of it.

Scripted drawer opening and closing. Our scripted policy for drawer opening and closing moves the gripper to the handle, then pulls or pushes it to open/close the drawer. At each step, Gaussian noise is added to the data collection and it does not succeed 70% of the times.

Pesudocode and more details of these policies, which are directly used from prior work [3] is provided in Algorithms 1-3 of Singh et al. [3].

D.2 Real-World Domains

Sawyer tasks. As detailed in Section D.2, the dataset used for our Sawyer tasks is the same as Khazatsky et al. [30]. We emphasize that we directly utilize the previously collected datasets from Khazatsky et al. [30] to mimic the real-world use case of offline RL, where we are supposed to learn effective policies from a previously collected dataset. The datasets for each of two tasks (put lid on pot, open a drawer) consist of 100 trajectories which were collected using a 3Dconnexion SpaceMouse device. Each trajectory in both the datasets is of length 80, which is also the number of time steps provided to the learned policy for solving the task. We then label the trajectories using 0-1 reward indicating a success when the task is complete (i.e., the lid is on the pot, and the drawer is sufficiently open). We present some examples of trajectories in the dataset on the associated supplementary website https://sites.google.com/view/offline-rl-workflow.

Real WidowX Pick and Place task. We collect data for this task by utilizing a scripted policy that first localizes the object, then reaches for this object using noisy actions and then attempts a grasp (with added noise) and places the object on the tray imperfectly. The success rate of the policy is 35% in both the grasping and the placing of the object on the the tray. A reward of +1 was provided when the policy was able to place the object in the tray. Each trajectory in this dataset is of length 15, which is also the time-limit provided to the learned policy for solving the task at evaluation time. We provide videos of sample trajectories in the dataset in the associated supplementary website https://sites.google.com/view/offline-rl-workflow.

E Detailed Empirical Results

In this section, we provide additional empirical results for various components of our workflow, including missing evidence from the main paper.

E.1 Simulated Domains

Addressing overfitting in the grasping from blocked drawer task in Scenario #1. We first discuss the efficacy of the proposed correction for overfitting via the variational information bottleneck regularizer (Equation 3) on the grasping from blocked drawer task. As shown in Figure 12, utilizing the bottleneck regularizer gives rise to a stable trend in Q-values (Q-values no more decrease with more training) as shown in the blue curve compared to the orange curve for base CQL, and as is evident from the policy performance plot, utilizing our fix for overfitting also leads to higher and stable performance.

Scenario #2, multiple object pick and place task. We provide the details (loss curves and Q-value trends) for this task on our anonymous project website linked here: https://sites.google.com/view/offline-rl-workflow.

E.2 Real-World Experiments

Sawyer tasks. We present the missing CQL regularizer ($\mathcal{R}(\theta)$) plot for this task from the main text (Section 6) below. Note that even the CQL regularizer eventually increases (dashed lines in the figure below) in the case of the base CQL algorithm that does not utilize a large ResNet architecture. On the other hand, utilizing the ResNet architecture leads to a clearly decreasing trend in the value of the CQL regularizer as is evident below. Thus, utilizing a larger network addresses underfitting.

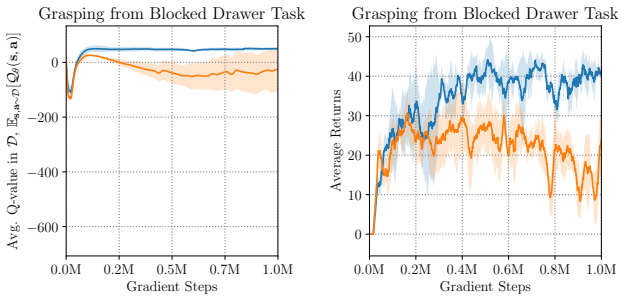
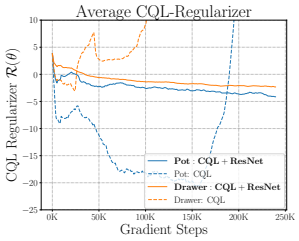


Figure 12: Trend in average dataset Q-value (left) and the performance of the policy (right) for base CQL (**orange**) and base CQL + overfitting correction using VIB (Equation 3) (**blue**). Note that using the VIB regularizer addresses overfitting in that the Q-values increase and then stabilize and this stabilization effect is also observed in the performance of the policy, which also increases around two fold.



More results and videos for each task can be found on our anonymous website located here: https://sites.google.com/view/offline-rl-workflow.

F Applying Our Workflow to Other Offline RL Algorithms

In this section, we discuss how to apply our workflow to other offline RL algorithms beyond CQL. Our workflow is applicable to conservative offline RL algorithms that can be interpreted as abstractly optimizing the objective in Equation 2 in some form. We elaborate on this class of algorithms in the next section (Appendix F.1) and then present, in Appendix F.2, an application of our workflow for detecting and correcting overfitting with BRAC [16], a policy-constraint conservative offline RL method.

F.1 Which Algorithms Does Our Workflow Apply To?

Our workflow guidelines are intended to be applicable to "conservative offline RL algorithms" that can be abstractly represented using the policy optimization objective shown in Equation 2, which is restated below for convenience of the reader:

$$\pi^* := \arg\max_{\pi} \; J_{\mathcal{D}}(\pi) - \alpha D(\pi, \pi_\beta) \qquad \text{(Conservative offline RL).} \tag{8}$$

$D(\pi, \pi_\beta)$ in Equation 8 represents the divergence between the learned policy $\pi(\cdot|\mathbf{s})$ and the behavior policy $\pi_\beta(\cdot|\mathbf{s})$ averaged over the state-visitation distribution of the learned policy π. This is given by $D(\pi, \pi_\beta) = \mathbb{E}_{\mathbf{s},\mathbf{a}\sim d^\pi_{\mathcal{D}}(\mathbf{s})\pi(\mathbf{a}|\mathbf{s})}\left[D(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s}))\right]$. Thus, Equation 8 can be expressed as:

$$J_{\mathcal{D}}(\pi) - \alpha D(\pi, \pi_\beta) = \mathbb{E}_{\mathbf{s},\mathbf{a}\sim d^\pi_{\mathcal{D}}}\left[\underbrace{r(\mathbf{s},\mathbf{a}) - \alpha D\left(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s})\right)}_{\text{effective new reward function}}\right]$$

This can be viewed as solving the RL problem with a modified reward function that penalizes deviation between the learned policy π and the behavior policy π_β. Thus, optimizing the policy against Equation 8 requires utilizing the long-term, cumulative estimate of divergence D.

Which algorithms are covered by our definition of conservative offline RL from Equation 8? Two algorithms covered under this definition are BRAC-v [16] and CQL [2]. While CQL ap-

plies a Q-function regularizer (Equation 1) to learn a conservative Q-function that directly models the combined effect of environment reward $r(\mathbf{s}, \mathbf{a})$ and divergence from the behavior policy $D(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s}))$ in the learned Q-function, BRAC-v instead exploits an explicit policy constraint. We discuss BRAC in detail below and demonstrate how to effectively apply our workflow to tune overfiting in BRAC in the next section.

Details and background on BRAC. Unlike CQL, BRAC-v explicitly subtracts the divergence $D(\pi(\cdot|\mathbf{s}'), \pi_\beta(\cdot|\mathbf{s}'))$ from the target value while performing the Bellman update. Additionally, since the divergence between the learned policy and the behavior policy *at the current state* is not a part of the Q-function, BRAC-v also explicitly adds the divergence value at the current state to the policy update. We instantiate the version of BRAC that uses the KL-divergence:

$$D(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s})) = D_{\mathrm{KL}}(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s})) = \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}\left[\log \pi(\mathbf{a}|\mathbf{s}) - \log \pi_\beta(\mathbf{a}|\mathbf{s})\right].$$

The first term in this divergence D_{KL} corresponds to an entropy regularizer on the policy $\pi(\cdot|\mathbf{s})$ that standard MaxEnt RL algorithms like Soft Actor-Critic (SAC) [65] already apply. To estimate the second term, BRAC estimates a model of the behavior policy, that we denote as $\hat{\pi}_\beta$, and uses it to explicitly compute this divergence. Denoting the policy and the Q-function as π_ϕ and Q_θ, the BRAC-v training objectives are (practical implementations use different values for α and β):

$$\text{Q-function:} \quad \min_\theta \; \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}\left[\left(r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{\mathbf{a}'\sim\pi_\phi(\cdot|\mathbf{s}')}[\bar{Q}_\theta(\mathbf{s}',\mathbf{a}') + \beta \log \hat{\pi}_\beta(\mathbf{a}'|\mathbf{s}')] - Q_\theta(\mathbf{s},\mathbf{a})\right)^2\right].$$

$$\text{Policy:} \quad \max_\phi \; \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi_\phi(\cdot|\mathbf{s})}\left[\underbrace{Q_\theta(\mathbf{s},\mathbf{a}) + \beta \log \hat{\pi}_\beta(\mathbf{a}|\mathbf{s})}_{\text{conservative Q-value; } Q_c(\mathbf{s},\mathbf{a})} - \underbrace{\alpha \log \pi_\phi(\mathbf{a}|\mathbf{s})}_{\text{policy entropy; standard MaxEnt RL}}\right]. \tag{9}$$

We will refer to the estimate $Q_c(\mathbf{s},\mathbf{a}) := Q_\theta(\mathbf{s},\mathbf{a}) + \beta \log \hat{\pi}_\beta(\mathbf{a}|\mathbf{s})$ as the *conservative Q-value*, that estimates the combined effect of both the reward and the divergence from the behavior policy. $Q_c(\mathbf{s},\mathbf{a})$ is analogous to the Q-function trained via CQL which directly estimates this combined effect. To note further similarities, observe that CQL optimizes the policy against the conservative Q-value estimate, predicted directly by the Q-network, along with an added entropy regularizer, whereas BRAC uses Q_c (Equation 9) in its place. Q_c will play a crucial role in adapting our workflow for overfitting to BRAC which we discuss in the next section.

Which offline RL methods is our workflow not applicable to? The formulation of conservative offline RL in Equations 2 and 8 does not encompass offline RL algorithms that only utilize a "myopic" behavior regularization, such as BCQ [52], BEAR [53], AWR [66], TD3+BC [67]. These methods only apply the behavior constraint locally at the current state and do not propagate its effect through the Bellman backup. The Q-functions for such myopic behavior-regularized algorithms are trained in a similar fashion as standard online actor-critic algorithms, and so we would not expect the Q-values of these algorithms to exhibit similar trends as conservative Q-functions. Our proposed workflow is not designed to handle such methods, though extending it to address them is an interesting direction for future work.

F.2 Empirical Demonstration: Applying Our Overfitting Workflow to BRAC

In this section, we adapt our proposed workflow (Metrics 3.1 and Guidelines 3.1 and 4.1) for detecting and addressing overfitting to the behavior-regularized actor-critic (BRAC) algorithm and empirically verify the efficacy of these metrics and guidelines. Per the discussion above, the main modification needed to apply our workflow from CQL to BRAC is to utilize the conservative Q-value estimate $Q_c(\mathbf{s},\mathbf{a})$ for BRAC, instead of the Q-values estimated by the Q-network which worked in the case of CQL. Barring this modification, the key principles of our workflow remain the same for BRAC. We detail these below, and present a comparison against our workflow for CQL in Table 3.

Detecting overfitting in BRAC. Unlike CQL, where the learned Q-values represent a conservative Q-estimate that accounts for both the reward and the divergence from the behavior policy, BRAC estimates these quantities separately as shown in Equation 9, with the Q-value not accounting for the divergence against the behavior policy at the current state. Therefore, to apply our workflow guidelines (Metric 3.1, Guideline 3.1) to BRAC, we track the "conservative Q-value estimate" discussed in the previous section ($Q_c(s,a) := Q_\theta(s,a) + \beta \log \hat{\pi}_\beta(a|s)$), which is BRAC's analogue

Metric/Guideline	CQL (Main paper)	BRAC-v (Appendix F.2)
Metric 3.1 (Detecting overfitting)	Low average dataset Q-value, $\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q_\theta(\mathbf{s},\mathbf{a})]$, decreasing with more gradient steps	Low average conservative Q-value, $\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q_c(\mathbf{s},\mathbf{a})]$ on the dataset, that is decreasing with more gradient steps
Guideline 3.1 (Policy selection)	If overfitting is detected, select the checkpoint with highest average dataset Q-value before overfitting	If overfitting is detected, select the checkpoint with highest average dataset conservative Q-value before overfitting
Guideline 4.1 (Addressing overfitting)	Use some form of capacity-decreasing regularizer on the Q-function, e.g., VIB regularizer, Dropout, etc	Use some form of capacity-decreasing regularizer on both the estimated behavior policy $\hat{\pi}_\beta$ and Q-function $Q_\theta(\mathbf{s},\mathbf{a})$, since both combine to form the conservative estimate Q_c

Table 3: Summary of how our proposed overfitting workflow for CQL in the main paper can be adapted to BRAC, with main modifications from CQL to BRAC highlighted in red. The primary modification is to utilize conservative Q-value estimates, $Q_c(\mathbf{s},\mathbf{a})$ for BRAC (Equation 9), instead of the outputs of the Q-network.

of the Q-value learned by CQL. Similar to CQL, overfitting in BRAC-v can be detected via a non-monotonic trend in average dataset conservative Q-value: if the average dataset conservative Q-value first increases and then decreases with more training, this indicates the presence of overfitting. We summarize this in Table 3, first row.

Policy selection for BRAC. When overfitting is detected, i.e., the conservative Q-value estimates first increase and then decrease with more gradient steps, we utilize early stopping to find a good policy checkpoint within this run for deployment. Analogously to CQL, our policy checkpoint selection guideline (Table 3, second row) suggests that a good checkpoint can be found by picking the one that attains the highest average conservative Q-value on the dataset before overfitting begins.

To empirically verify if the adaptation of our workflow is effective for BRAC-v, we ran experiments on the simulated grasping from drawer task from Scenario #1, with offline datasets containing 100 and 200 trajectories. Observe in Figure 13 (left), that with 100 trajectories, the average dataset conservative Q-values $\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q_c(\mathbf{s},\mathbf{a})]$ first increases and then drops with more gradient steps. This observation is consistent with what we expect to happen if the run of BRAC-v overfits per the guideline in Table 3. In the figure, the vertical dashed lines indicate the policy checkpoints that will be selected by our policy selection guideline (Table 3). We also visualize the performance of the chosen checkpoints against the actual policy return in the top row for analysis purposes. Note that the selected policy checkpoint indeed attains close to the peak performance achieved over the entire training run of BRAC-v. This indicates the efficacy of our workflow for detecting overfitting and performing policy selection for the BRAC-v algorithm.

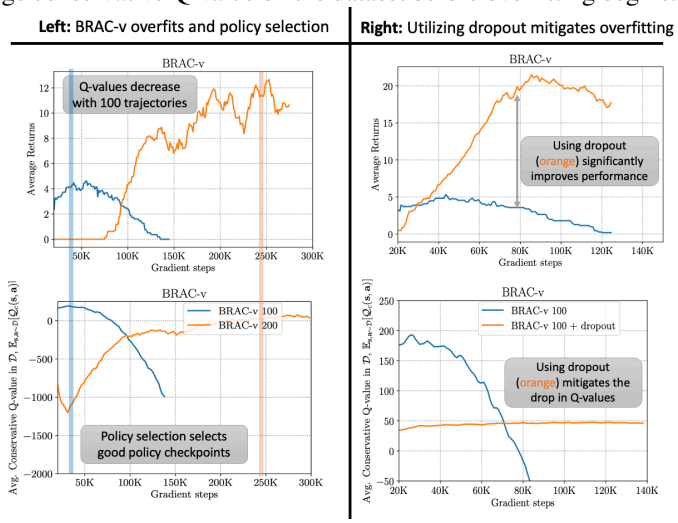


Figure 13: **Left: Overfiting and policy selection for BRAC-v**: Policy performance **(top)** and average dataset conservative Q-value **(bottom)** with varying number of trajectories (100 and 200). The conservative Q-value for the run with 100 trajectories (blue) eventually decreases, while it is relatively stable for 200 trajectories (orange). Vertical bands indicate regions around the peak Q-value and observe that these regions correspond to policies with good actual performance. **Right: Addressing overfitting in BRAC-v** by using the capacity-decreasing dropout regularizer leads to stable and non-decreasing conservative Q-values and improved policy performance.

Addressing overfitting in BRAC-v. Once overfitting is detected, we need to find an method to alleviate it. As in our workflow for CQL, we can add any capacity-decreasing regularizer such as dropout [24], variational information bottleneck (VIB), etc to mitigate overfitting. Technically, we want to apply this regularization on the conservative Q-function estimator, $Q_c(\mathbf{s}, \mathbf{a})$, but in the case of BRAC-v, this quantity is not estimated using a single neural network. Thus, we recommend applying the capacity-decreasing regularization to both the critic ($Q_\theta(\mathbf{s}, \mathbf{a})$) and the behavior policy estimate $\hat{\pi}_\beta(\cdot|\mathbf{s})$ separately. This guideline is summarized in the third row of Table 3.

To empirically validate our guideline for addressing overfitting in BRAC-v, we applied the capacity-decreasing dropout regularizer on the run of BRAC on the grasping from drawer task with 100 trajectories. We chose the dropout regularizer since it worked for CQL (Figure 19, Appendix J), and because it is easier to apply than two separate information bottlenecks on the Q-function and the estimated behavior policy. As shown in Figure 13 (right), applying dropout not only alleviates the drop in conservative Q-value estimates after many gradient steps, but it also allows us to pick later checkpoints in training, all of which perform equally well, and much better than the base BRAC-v algorithm. Crucially note that while the policy performance of BRAC-v degrades to zero with more training, utilizing dropout improves the policy performance and increases stability. This validates that overfitting in BRAC-v, as detected via our workflow, can be effectively mitigated by decreasing the capacity of the conservative Q-function in BRAC, in this case by applying dropout to the Q-network and the estimated behavior policy.

G What About the Hyperparameter α?

The guidelines in the preceding paragraph suggest how to adjust capacity, but do not tell us how to tune the multiplier on the CQL term, α, in Equation 1. This multiplier trades off minimizing TD error with a correction for distributional shift. An inappropriate choice of α will inhibit good policy performance, since CQL would be insufficiently constrained against out-of-distribution actions with excessively low values of α, while being too constrained to stay close to the dataset with excessively high values. In our experiments, both in simulation and in the real-world, we found that a default value of $\alpha = 1.0$ taken from prior work [3] worked for all scenarios without any tuning; however, we do provide guidelines for tuning α values if required. We expect that tuning α is especially needed when the data is highly diverse or when it is generated from a narrow expert policy.

How can we detect excessively large α values? Since a larger value of α would correspond to a higher weight on the CQL regularizer $\mathcal{R}(\theta)$, which minimizes Q-values, we would expect that Q-values learned with a large α would exhibit an overfitting trend per Metric 3.1, where Q-values would decrease with more training steps. Thus, if the Q-values on the dataset exhibit a decreasing (overfitting-like) trend despite applying the mitigation strategies in Section 4, it indicates that α may be too large and we need to reduce α. This is formalized as:

Guideline G.1 (Guideline for decreasing α)**.** *If a run of CQL with $\alpha = \alpha_0$ exhibits a trend that resembles overfitting per Metric 3.1 and correcting for overfitting based on Guideline 4.1 does not address it, then re-run CQL with a smaller value of $\alpha = \alpha_1$. If this new run with $\alpha = \alpha_1$ exhibits overfitting as well, decrease α from α_0 and α_1.*

How can we detect excessively small α values? When α is too small, we would expect that the Q-values do not decrease with more training, since the CQL regularizer has minimal effect. Thus a run of CQL with a very small α will resemble underfitting, as identified by Metric 3.2. Given a run with non-decreasing Q-values and a high value of the training CQL regularizer, our first step is to determine if the run is underfitting due to insufficient capacity or just has a smaller α. Thus, we first detect underfitting (Metric 3.2) and re-run training with a higher-capacity model (e.g., a Resnet policy, DR3 [22] capacity-increasing regularizer). If we find that even higher-capacity models are unable to reduce the value of the CQL regularizer during training, then this indicates that α is too small. This is expected because, no matter what the capacity of the model, a small α would cause the policy to pick unseen, out-of-distribution actions due to erroneous Q-function overestimation. Once such a scenario is detected, we can increase α, until the value of the CQL regularizer is sufficiently negative and then utilize the other workflow guidelines.

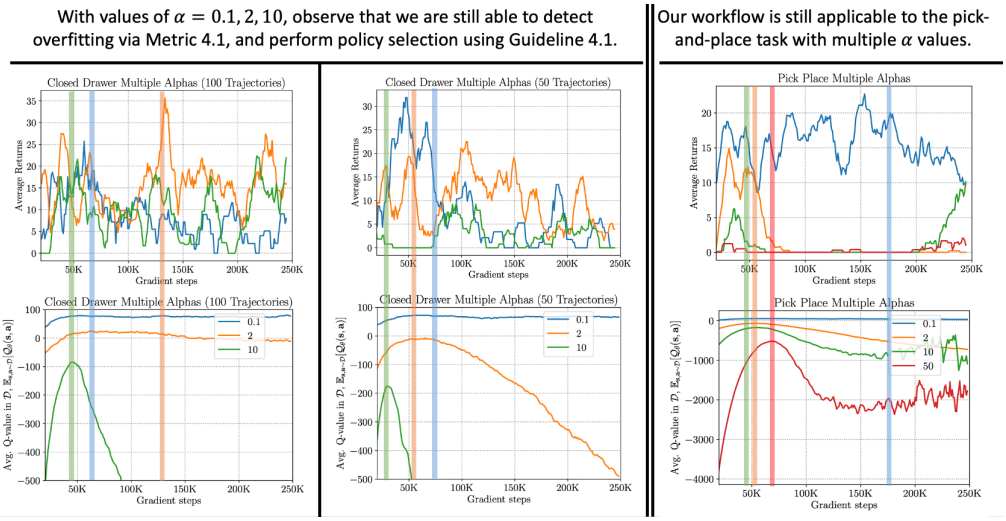


Figure 14: **Evaluating our overfitting workflow with multiple values of the CQL hyperparameter** $\alpha \in \{0.1, 2, 10, 50\}$ on three tasks from Scenario #1: grasping from drawer task with 50 and 100 trajectories and the pick-and-place task with 100 trajectories. Observe that with all of these values, the average dataset Q-value first increases and then decreases, which indicates the presence of overfitting. Also note that policy checkpoints prescribed by our policy selection guideline perform well, especially when compared to other checkpoints within the run. The performance of both CQL and our workflow is generally poor in runs with large $\alpha = 10$, because large α values constrain the learned policy to be close to the behavior policy and our workflow does not improve the policy performance in this case. We additionally evaluate $\alpha = 50.0$ for the pick-and-place task and observe that the run is overfitting, however, the policy performance is bad for all the checkpoints because α is too large, making CQL similar to behavior cloning.

Guideline G.2 (Guideline for increasing α)**.** *If a run of CQL exhibits a trend that resembles underfitting per Metric 3.2, and increasing model capacity per recommendations mentioned in Guideline 4.2 does not reduce the CQL regularizer, then we suggest first increasing the coefficient of the CQL regularizer α until the final value of the CQL regularizer is lower than 0, and then applying the other workflow guidelines with this new α value.*

H Experiments Tuning The CQL α Hyperparameter

In our experiments on both simulated domains and real robots, we utilized a default value of $\alpha = 1.0$ as the multiplier on the CQL term. This directly follows from the choice made in prior work [3], without any modification or tuning. However, to understand the effect of α, we now evaluate our workflow on runs with various α values, $\alpha \in \{0.0, 0.01, 0.1, 2, 10, 50\}$, using the two tasks (pick-and-place task and grasping from drawer task) from Scenario #1 with 50 and 100 trajectories. Generally, we find that our workflow for detecting overfitting and performing policy selection is reasonably effective for a range of values with $10 \geq \alpha \geq 0.1$, but fails to learn a good policy with very low α values (≤ 0.01), for which CQL does not prevent catastrophic overestimation. Similarly our workflow is unable to improve the policy performance in CQL runs with very high α values, which lead to Q-functions that overwhelmingly prioritize giving high value to dataset actions and lead to imitation-like behavior. It is therefore necessary to avoid such extreme α values. In this section, we apply our proposed guidelines for detecting if α is too small or too large (Guidelines G.1 and G.2) and first adjust α. We first discuss $\alpha \geq 0.1$, and then the lower values.

H.1 Values of α That Are Not Too Small

We present the trend in average dataset Q-values in Figure 14 for $\alpha \geq 0.1$. Observe that for $\alpha \in \{0.1, 2, 10\}$, the average dataset Q-value first increases and then decreases with more gradient steps, indicating the presence of overfitting per Metric 3.1. Since overfitting is detected, we can perform policy selection using Guideline 3.1 by choosing the policy checkpoint that appears near the peak in the average dataset Q-value for deployment. For each α, these checkpoints are marked with a

vertical dashed line. Observe that the selected policy checkpoint indeed performs well compared to all other checkpoints within each training run. This indicates that Metric 3.1 and our policy selection rule in Guideline 3.1 work well across these α values. However, the performance of CQL with large α values is worse compared to smaller α values, likely because CQL finds a policy close to the behavior policy when the α values are too large (e.g., $\alpha = 50$ for the pick-place task in Figure 14, or $\alpha = 10$ for the drawer task with 50 trajectories in Figure 14). Thus, no matter how we select the policy checkpoint, the performance would be bad, since no checkpoint in the run actually attains good performance. We will shortly discuss how we can detect if α is large and decrease it, but we first present results of applying the VIB overfitting correction to runs with various α values.

Since overfitting is detected, we would attempt alleviate overfitting by utilizing the VIB regularizer (Equation 3) following Guideline 4.1. As shown in Figure 15, the VIB regularizer leads to improved policy performance for $\alpha \in \{0.1, 2\}$. However, the VIB regularizer is ineffective with $\alpha = 10.0$, where it does not improve performance.

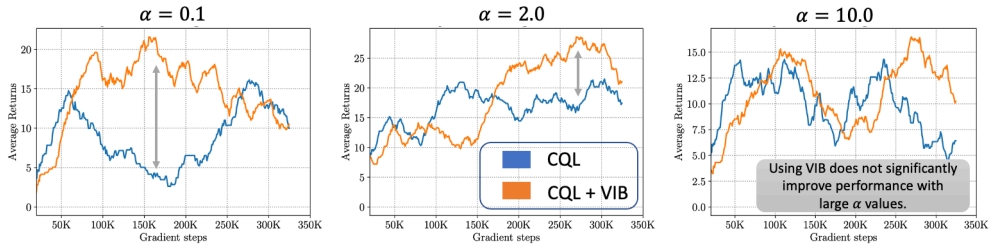


Figure 15: **Utilizing the VIB regularizer from Equation 3 to correct overfitting in CQL runs with** $\alpha \in \{0.1, 2, 10\}$ for the grasping from drawer task with 100 trajectories. Applying the VIB regularizer to decrease capacity and correct overfitting improves performance with $\alpha = 0.1$ and $\alpha = 2.0$, but does not improve performance with a larger value of $\alpha = 10.0$.

The above evidence indicates that our overfitting workflow can fail if the α value is too large ($\alpha \geq 10.0$), but our workflow improves the performance of CQL when $\alpha \in \{0.1, 1.0, 2.0\}$. Hence, for large αs we first follow Guidelein G.1 to decrease α before applying our overfitting workflow.

To validate the efficacy of Guideline G.1, we point the reader to Figure 14. If we start with $\alpha = 10.0$ or 50.0 on the the drawer task with 50 trajectories or the pick-and-place task, Guideline G.1 would prescribe reducing α, since smaller α values such as $\alpha = 0.1, 1.0, 2.0$ also exhibit overfitting per Metric 3.1. Doing so also does improve the policy performance, especially when starting from $\alpha = 50.0$, indicating that this guideline is effective.

H.2 Small values of α

Next, we evaluate our workflow with the two smallest values of $\alpha = 0.0, 0.01$. In both cases, as shown in Figure 16, we find that the value of the CQL regularizer is large (close to 0, which means out-of-distribution actions have similar values as in-distribution actions), and average dataset Q-value does not decrease with more gradient steps. As expected, the corresponding policy performs poorly in each case, since a high CQL regularizer value implies that the out-of-distribution actions have a higher Q-value than in-distribution actions, which in turn means that policy optimization will select out-of-distribution actions. In this case, our underfitting workflow will not improve the performance of CQL, and the value of α would need to be raised. We thus follow Guideline G.2 first, to tune α before applying the rest of our workflow.

To empirically demonstrate the efficacy of Guideline G.2, we attempt to correct the apparent underfitting in the run of CQL with $\alpha = 0.01$ by rerunning it with increased model-capacity and present the results in Figure 17. Observe that the value of the CQL regularizer is still close to 0, identical to the the naïve CQL run without the underfitting correction. Since underfitting correction does not reduce the value of the CQL regularizer, according to Guideline G.2, we need to increase α to allow for better minimization of the CQL regularizer. As we have already seen in the earlier runs in this section in Figure 14, if we increase α to 0.1 or 1.0, the value of the CQL loss would be small and sufficiently negative attaining values around -5.0, and overfitting is detected. Our policy selection guideline would then allow us to find a good policy for deployment.

To summarize, while our workflow for detecting overfitting, performing policy selection, and correcting overfitting works well across several α values, CQL can fail when α is too small, and will reduce to behavior cloning when α is too large. This is expected because smaller α values are in-

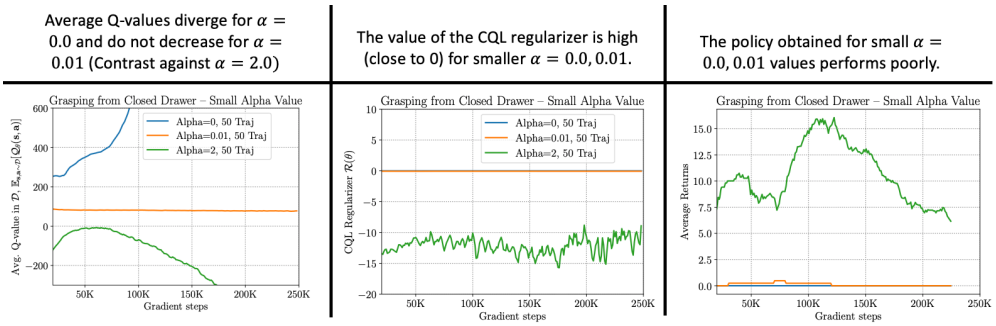


Figure 16: **CQL fails to prevent erroneous overestimation in the Q-function with small α values and hence performs poorly.** For $\alpha = 0.0$, the Q-function positively diverges. For $\alpha = 0.01$, the average Q-value is stable and does not decrease with more gradient steps. Note the vastly different trend in the average Q-value for $\alpha = 2.0$ for contrast. Additionally, observe that the value of the CQL regularizer is close to 0 for $\alpha = 0.0$ and $\alpha = 0.01$, which means Q-values for out-of-distribution actions are high compared to in-distribution actions, fo

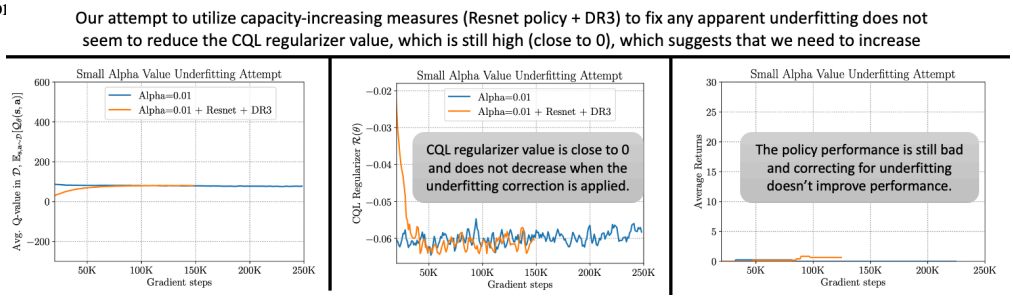


Figure 17: **Validating Guideline G.2 by attempting to fix underfitting in CQL with a small $\alpha = 0.01$.** To verify if the run of CQL with $\alpha = 0.01$ is underfitting or if it requires increasing α, we re-run it with a capacity-increasing measures. However, even in this case, the value of the CQL regularizer is large. The value of the CQL regularizer is close to 0, which means the values of out-of-distribution actions is not small enough compared to in-distribution actions. Since underfitting correction does not help in this case, we conclude that is the case where the value of α needs to be raised to obtain improved performance.

sufficient to prevent overestimation and will cause the policy to choose unseen out-of-distribution actions and extremely large α values will strongly update the policy towards the behavior policy. To detect and handle if α is too small or too large, we proposed Guidelines G.1 and G.2, which prescribe **increasing** α if **(a)** the value of the CQL regularizer is large, and **(b)** utilizing capacity-increasing measures does not lead to reduction in the CQL regularizer value and **decreasing** α if **(a)** overfitting is observed with the current run, and **(b)** re-running CQL with a smaller value of α also exhibits an overfitting trend, with average Q-value decreasing with more gradient steps. After modifying the value of α, we prescribe following the recommendations of the rest of our workflow.

I Underfitting With 10 and 20 Objects in Scenario #2

In this section, we present our results on applying the proposed capacity-increasing measures on the simulated experiments with multiple training objects (10 and 20 objects) from Scenario #2. The plot for 35 objects is shown in the main paper. In the case of 10 and 20 objects, we also observed a high value of TD error (see Figure 5), with relatively stable Q-values. In this case, our workflow would prescribe correcting for underfitting. In Figure 18, we present results of running CQL with the capacity-increasing DR3 regularizer and a ResNet policy to address underfitting in the case of 10 and 20 objects. We find that in both cases the performance of the policy improves and our capacity-increasing measures also generally lead to a slight decrease in the value of the TD error.

J Other Capacity-Decreasing Regularizers for Addressing Overfitting

In this section, we present a study that evaluates different choices of capacity-decreasing regularizers to prevent overfitting in CQL. The candidate capacity-decreasing regularizers we evaluate are:

- dropout [24] with masking probability p on the layers of the Q-function Q_θ,

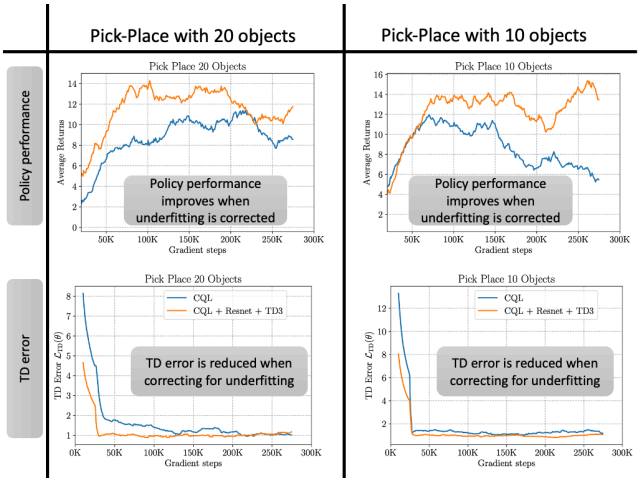


Figure 18: **Correcting underfitting by utilizing a ResNet policy + DR3 regularizer for the case of 10 and 20 objects from Scenario # 2.** Note that the addition of these underfitting corrections improves policy performance, while also reducing the training TD error by some amount.

- ℓ_1 regularization on the parameters θ of the Q-function (i.e., $\min_\theta \mathcal{L}_{\text{CQL}}(\theta) + \rho||\theta||_1$), and
- ℓ_2 regularization on the parameters θ (i.e., $\min_\theta \mathcal{L}_{\text{CQL}}(\theta) + \rho||\theta||_2^2$).

We apply each of these regularizers to the run of CQL on the pick-and-place task from Scenario #1, with 100 trajectories and report the average dataset Q-value, the corresponding performance of the policy (for analysis purposes) and the value of the training CQL regularizer for each of dropout, ℓ_1 and ℓ_2 regularization schemes in Figure 19. To find a good value of ρ and p completely offline, we run each regularizer with different values of the hyperparameter $\rho \in \{0.1, 0.01, 1.0\}$ (for ℓ_1/ℓ_2 regularization) and $p \in \{0.01, 0.1, 0.2, 0.4\}$ (for dropout) and pick the value that stabilizes the trend in the average dataset Q-value, while not inhibiting the minimization of the training CQL regularizer. That is, we require the value of CQL regularizer to be sufficiently negative (ideally ≤ -2 or -3). This is essential since excessive capacity-decreasing regularization can inhibit the minimization of the training CQL objective which would cause the policy to execute bad out-of-distribution actions. Using the scheme described above, we obtained $p = 0.2$ for dropout and $\rho = 0.01$ for the case of ℓ_2 regularization.

Observe in Figure 19 that utilizing dropout (left column) or applying ℓ_2 regularization (middle column) mitigates the drop in average Q-value that is observed with naïve, untuned CQL on this task while also achieving a small CQL regularizer value. Applying dropout and ℓ_2 regularization leads to improved and much more stable policy performance. This indicates that addressing overfitting by applying capacity-decreasing regularization can lead to improved performance.

We observed that ℓ_1 regularization did not give rise to improved performance. Out of all three values of ρ, all of which are presented in Figure 19 (right column) we found that $\rho = 0.01$ was likely not large enough to mitigate the drop in Q-value, and runs with larger values of $\rho = 0.1, 1.0$ failed to decrease the training CQL regularizer. We believe that an intermediate value of $\rho \in [0.01, 0.1]$ can possibly alleviate the overfitting issue, and we will run a finer search over ρ for the final version.

K Alternative Metrics for Overfitting

In addition to Metric 3.1 that prescribes tracking the average dataset Q-value for detecting overfitting and performing policy selection (Guideline 3.1), we can also, in principle, choose to use an estimate of the policy return estimated using the learned conservative Q-function. Formally, this metric is given by policy value averaged under the initial state distribution $\mu_0(\mathbf{s})$: $\mathbb{E}_{\mathbf{s}\sim\mu_0,\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[Q_\theta(\mathbf{s},\mathbf{a})]$. We perform a preliminary experimental study comparing metric $\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q_\theta(\mathbf{s},\mathbf{a})]$ (Metric 3.1) and the policy return at the initial state on the drawer and the pick-place tasks from Scenario #1, with 50 trajectories. As shown in Figure 20, we find that both of these metrics closely follow each other for most of the training iterations, and applying the policy selection guideline on either of them will choose the same policy checkpoint since these curves heavily overlap near the peak.

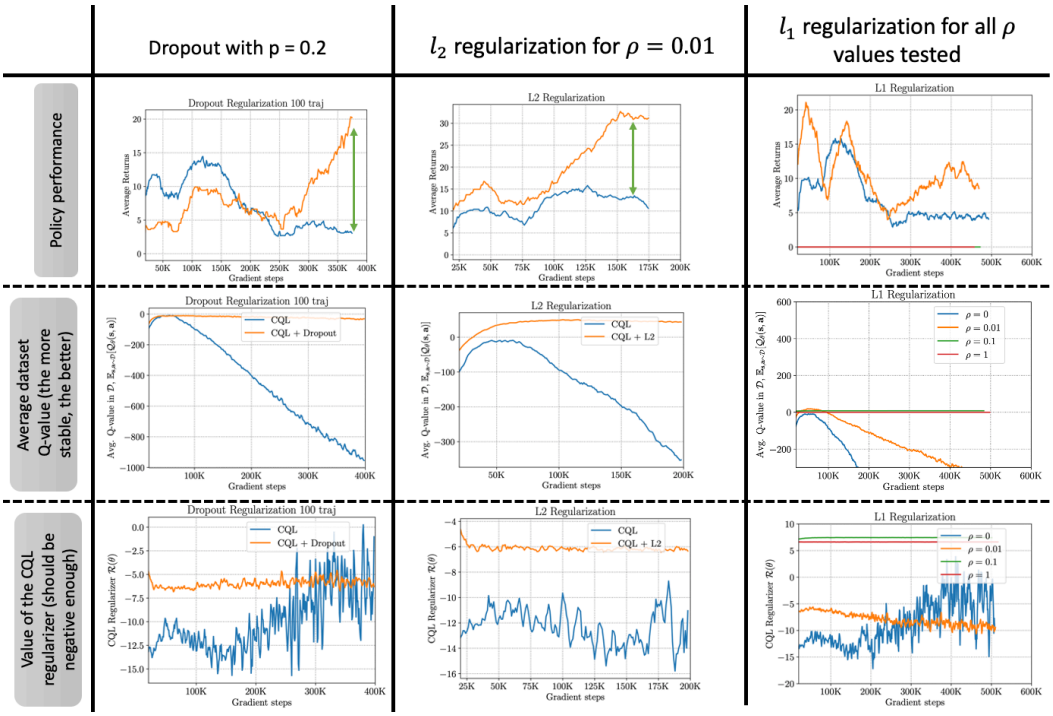


Figure 19: **CQL + different capacity-decreasing regularizers: dropout (left), ℓ_2 regularization (middle) and ℓ_1 regularization (right).** Comparison of different capacity-decreasing regularization schemes for the run of CQL on the drawer task with 100 trajectories from Scenario #1. While naïve CQL (shown in blue in the plots) exhibits overfitting, i.e., the average dataset Q-value first increases and then decreases with more gradient steps, the addition of ℓ_2 regularization or dropout with $\rho = 0.01$ and $p = 0.2$ respectively alleviates the drop in the Q-value (middle row shows the time series of the average dataset Q-value). Additionally observe that ℓ_2 regularization and dropout improve policy performance, especially ℓ_2 regularization. For ℓ_1 regularization, none of the ρ values we searched over was able to mitigate the drop in the average Q-value while retaining a negative value of the CQL regularizer, and thus did not improve in performance.

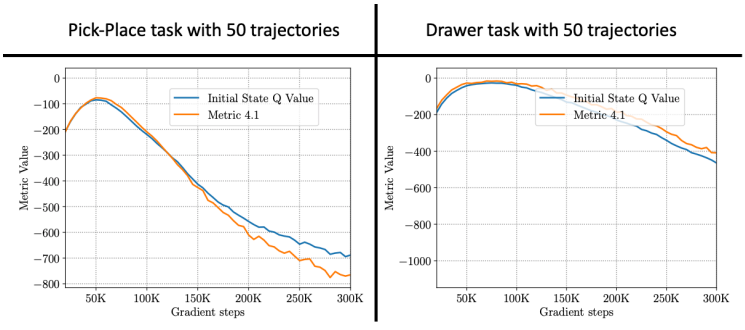


Figure 20: **Preliminary experiments comparing the evolution of the average dataset Q-value in Metric 3.1 (orange) and policy value averaged under the initial state distribution (blue).** Observe that both of these metrics follow each other closely for the most part in training, and exhibit a similar behavior, where the metric first increases and then decreases with more training. The peak in both of the curves overlap, indicating that utilizing either of the metrics for policy selection will return the same policy checkpoint.