# Appendices

## A Details of the Didactic Navigation Example from Section 3.1

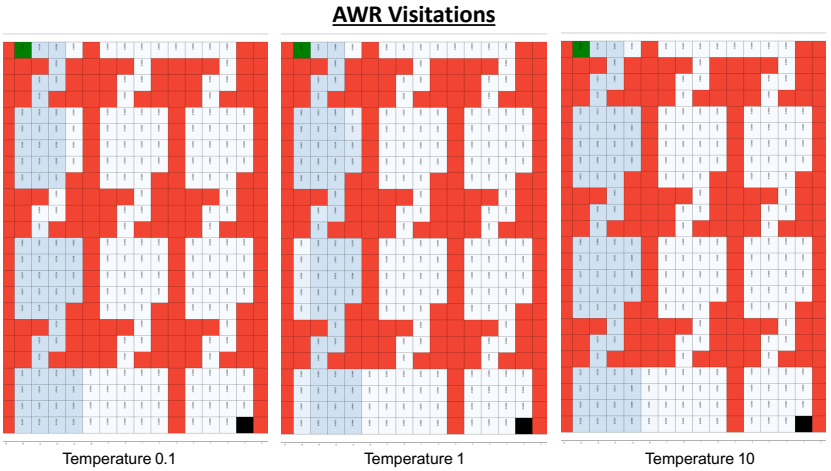


Figure 5: **Results of running AWR on the gridworld maze for a variety of temperature values** annotated with state-occupancy values. Observe that AWR is unable to reach the goal across a wide range of temperatures for the AWR hyperparameter. Even though for some of these hyperparameters, such as $\tau = 0.1$, it is able to traverse quickly into the third region, it does spend a larger fraction of its state visitation in the narrow hallways in this case (e.g., the final narrow hallway it is able to reach to) indicating that it gets stuck. Increasing the temperature to $\tau = 1.0$ does not solve it either, since now it does not even reach this hallway with as high of an occupancy.

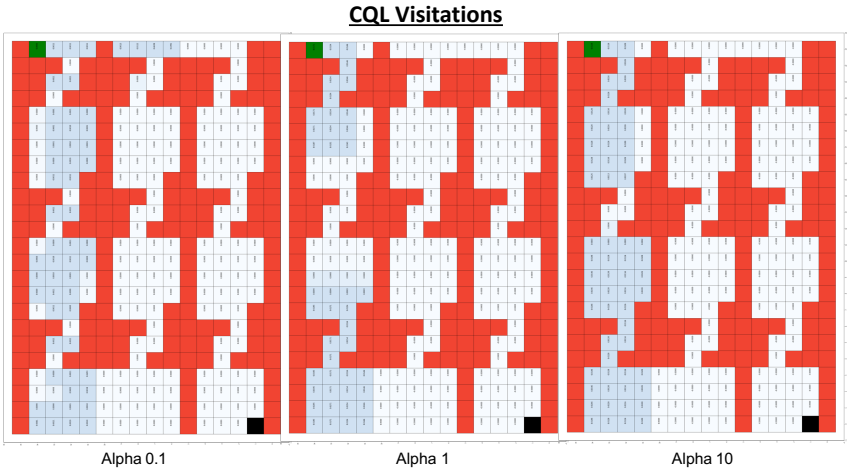


Figure 6: **Results of running CQL on the gridworld maze for a variety of temperature values.** Observe that CQL is unable to reach the goal and gets stuck for all the $\alpha$ values we studied.

In this section, we present some details regarding the navigation example we considered in Section 3.1. To create this example, we modified the gridworld code from Fu et al. (2019) (code taken from: https://github.com/justinjfu/diagnosing_qlearning) to create the corresponding gridworld maze. We utilize a $24 \times 16$ gridworld as shown in the figures below, larger than the $8 \times 8$ or $16 \times 16$ gridworlds studied in this repository in the past. We utilize a smooth representation of the observation space. This is constructed by first sampling random Gaussian feature vectors from $\mathbb{R}^{50}$ for each grid cell (each grid cell is a state). Smoothing of these features vectors is then done locally, following the protocol for "grid-*-smoothobs" in Fu et al. (2020). This observation type presents a challenge for Q-learning algorithms that may often generalize incorrectly. This is because aliasing occurs with the predictions of nearby states that exhibit very different dynamics

but not so different observation vectors. The agents gets a sparse binary 0/+1 reward: +1 is attained only when the agent reaches the goal (marked in black), starting from the start location (marked in green).

The behavior policy in each part of the maze is based on a mixture of different policies. In the wider rooms of the maze, one of the policies is a uniform policy, that uniformly chooses every action at a state. The second policy is a biased policy, that drives the agent away from the goal. In the narrow hallways, the behavior policy deterministically drives the agent towards the goal. For wide rooms, in contrast, a bias exists for actions taken in the direction away from the goal. This bias was set to be 0.8. This means that for rooms where you need to exit the wide passage by going down, the action of going up was selected 80% of the time by the behavior policy and each of the other action was randomly sampled with 20% probability. The opposite decision was taken for rooms where the goal was towards the top of the wide passage where 80% of the time the down action was selected.

**Result visualizations.** We now present some visualizations of the policies learned by various methods: AWR, CQL and CQL (ReDS). Since the reward values are binary and sparse, return curves for AWR and CQL are not as informative, since they attain a 0 return in any episode, even if they make some progress. Therefore, we present the results in the form of state-visitation density plots under rollouts from the learned policy. Observe in Figures 5 and 6 that neither AWR or CQL are able to actually successfully traverse the maze, and get stuck in it. Note that $\tau$ and $\alpha$ control the strength of the distributional constraint in the methods AWR and CQL respsectively. Varying the temperature hyperparameter $\tau$ for AWR and $\alpha$ for CQL does modify the density in the narrow hallways, but utilizing too small of a parameter leads the agent to more frequent crashes in the narrow hallways. This makes the agent spend a higher fraction of its visitation in one such narrow region, whereas a higher $\tau$ or $\alpha$ does not even reach the third narrow hallway with a high-enough visitation.

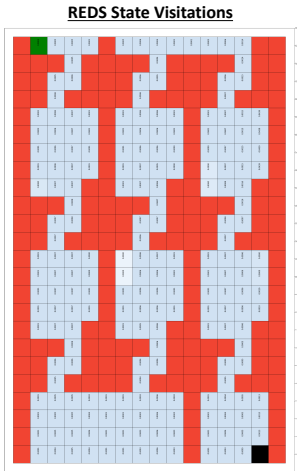


Figure 7: **Results of running CQL (ReDS) on the gridworld.** Note that CQL (ReDS) does actually succeed at solving the task.

On the other hand, the method we propose in this paper, ReDS, when combined with CQL is able to successfully traverse this maze, as shown in Figure 7.

**Analysis.** These experiments are consistent with our expectations. When distributional constraints are faced with highly heteroskedastic action distributions at subsequent states, they failed to perform good actions at these consecutive states. A strong distributional constraint leads to the agent being too close to the behavior policy, whereas a weaker constraint fails to identify good actions at states where the behavior policy is narrow.

### A.1 Why and When Do Distributional Constraints Fail in scenarios with Heteroskedastic Data?

In this section, we shall discuss why distributional constraints are especially worse than support constraints in scenarios with heteroskedastic data. Let's first consider a simple single-state bandit problem. In this simple one-state problem, we must find a single optimal action by using offline

data. At first, it might appear that when faced with a wide behavior policy, distributional constraints would clearly fail since they would not deviate far away from the behavior policy. However, note that by carefully choosing the strength of the distributional constraint, we can, in principle, control the strength of the distributional constraint quite effectively. To see this concretely, consider applying CQL (Equation 2) to a single-state bandit problem. The Q-values on actions not observed in the training dataset will be clearly pushed down to $-\infty$. In addition, choosing a non-zero $\alpha$ will allow us to precisely control how close the action taken by the learned policy is to the best in-support action vs how close the learned policy is to the behavior policy. Therefore, for an optimally chosen value of $\alpha$, distributional constraint methods such as CQL should already work well in the single-state setting.

However, the precise challenge with distributional constraints arises in the multi-state setting, where finding a single value of $\alpha$ that can work well at all states might be can be exceedingly challenging in practice. This is the setting with heteroskedastic data that our paper considers, including in our didactic example above. In such a setting, a strong distributional constraint is able to take a desirable action when the behavior policy is narrow (for example in the narrow room). However, the action distribution has wider support in the wider room and the strong distributional constraint would not take a desirable action in this setting due to the high entropy of the behavioral distribution in this setting. In contrast, with a weaker distributional constraint, a bad out-of-distribution action may be taken where the behavior policy is narrow which can lead to instability in the policy. Note that these challenges are not, however, present in the single-state scenario. Our method CQL (ReDS) can tackle the problem in this setting by modulating the strength of the constraint per state. This allows the agent to stay close to the behavior distribution in the narrow room as well as learning to output the most desirable in-support action in the wider room.

# B PROOFS

In this appendix, we will provide proofs for the various theoretical results in the main paper: Theorem 3.1. We will first discuss some preliminaries and notation, then present the proof for Theorem 3.1, and finally the remaining results.

## B.1 NOTATION AND PRELIMINARIES

Let $\pi_\beta(\mathbf{a}|\mathbf{s})$ denote the behavior policy. Note that the dataset, $\mathcal{D}_i$ is generated from the marginal state-action distribution of $\pi_\beta$, i.e., $\mathcal{D} \sim d^{\pi_\beta}(\mathbf{s})\pi_\beta(\mathbf{a}|\mathbf{s})$. Define $\widehat{d}^{\pi}$ as the state marginal distribution introduced by $\pi$ under the empirical MDP defined by the transitions in the dataset. Let $D_{\text{CQL}}(p, q)$ denote the following distance between two distributions $p(\mathbf{x})$ and $q(\mathbf{x})$ with equal support $\mathcal{X}$:

$$D_{\text{CQL}}(p, q) := \sum_{\mathbf{x} \in \mathcal{X}} p(\mathbf{x}) \left( \frac{p(\mathbf{x})}{q(\mathbf{x})} - 1 \right).$$

We drop the subsrcipt "CQL" from $D_{\text{CQL}}$ for clarity. (Kumar et al., 2020) showed that when optimizing the generic distributional constraint algorithm shown in Equation 1, the resulting policy $\pi^*$ attains a high probability safe-policy improvement guarantee, i.e., $J(\pi^*) \geq J(\pi_\beta) - \zeta$, where $\zeta$ is:

$$\zeta = \mathcal{O}\left( \frac{1}{(1-\gamma)^2} \right) \mathbb{E}_{\mathbf{s} \sim \widehat{d}^{\pi^*}} \left[ \sqrt{\frac{D(\pi^*, \pi_\beta)(\mathbf{s}) + 1}{|\mathcal{D}(\mathbf{s})|}} \right] + \frac{\alpha}{1-\gamma} \mathbb{E}_{\mathbf{s} \sim \widehat{d}^{\pi}} [D(\pi^*, \pi_\beta)(\mathbf{s})]. \quad (13)$$

We can further express $|\mathcal{D}(\mathbf{s})| = |\mathcal{D}||\mu(\mathbf{s})|$. The first term in Equation 13 corresponds to the decrease in performance due to sampling error and this term is high when the learned policy $\pi^*$ visits low density states under the dataset distribution (i.e., $\mu(\mathbf{s})$ is small) and when the divergence from the behavior policy $\pi_\beta$ is higher under these states. We will use this safe policy improvement guarantee in our proofs.

## B.2 PROOF OF THEOREM 3.1

In order to prove this result, we we will utilize a basic algebraic inequality mentioned below in Lemma B.1.

**Lemma B.1** (Algebraic variation)**.** *Given any $N$ positive real numbers, $x_1, x_2, \cdots, x_N$:*

$$\left(\sum_{i=1}^{N} \sqrt{x_i}\right)^2 \geq \sum_{i=1}^{N} x_i \geq \frac{1}{(N-1)} \sum_{i<j} (\sqrt{x_i} - \sqrt{x_j})^2. \tag{14}$$

*Proof.* For every $x_i$, define $y_i = \sqrt{x_i}$. Then, the difference between the two sides in the equation above is given by:

$$\sum_i y_i^2 - \frac{1}{(N-1)} \sum_{i<j} (y_i^2 + y_j^2 - 2y_i y_j) = \sum_i y_i^2 - \frac{N-1}{N-1} y_i^2 + \frac{1}{N-1} \sum_{i<j} 2y_i y_j \tag{15}$$

$$= \frac{1}{N-1} \sum_{i<j} 2y_i y_j \geq 0, \tag{16}$$

where the first step follows by rearranging $y_i$ and $y_j$, and the final step follows by noting that $y_i \geq 0$ for all $i$. The other inequality follows trivially by noting that $\sqrt{x_i}$ are positive, and applying the standard formula for sum of squares. □

We will also require a Lemma that allows us to upper bound the performance difference $J(\pi) - J(\pi_\beta)$ in terms of the metric $D_{\text{CQL}}(\pi, \pi_\beta)$ that appears in the safe policy improvement guarantee in Equation 13.

**Lemma B.2** (Tight upper bound on policy improvement.)**.** *Assume that the reward function $r(\mathbf{s}, \mathbf{a})$ of the MDP is bounded such that $\forall \mathbf{s}, \mathbf{a}, r(\mathbf{s}, \mathbf{a}) \in [-R_{\max}, R_{\max}]$. For any two policies $\pi$ and $\pi_\beta$, we have the following:*

$$J(\pi) - J(\pi_\beta) \lesssim \mathcal{O}\left(\frac{1}{(1-\gamma)^2}\right) \cdot \mathbb{E}_{\mathbf{s} \sim d^\pi} \left[D(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s}))\right] \cdot R_{\max}. \tag{17}$$

*Proof.* The core of the proof of this lemma relies on the fact that for any given function $\nu(\mathbf{x})$ over some space $\mathbf{x}$, we can upper bound, $\Delta_\nu(p, q) := \mathbb{E}_{\mathbf{x} \sim p}[\nu(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim q}[\nu(\mathbf{x})]$ in terms of $D(p, q)$. To show this, we expand this expression:

$$\Delta_\nu(p, q) := \sum_{\mathbf{x}} (p(\mathbf{x}) - q(\mathbf{x})) \cdot \nu(\mathbf{x}) \tag{18}$$

$$= \sum_{\mathbf{x}} q(\mathbf{x}) \cdot \left(\frac{p(\mathbf{x})}{q(\mathbf{x})} - 1\right) \cdot \nu(\mathbf{x}) \tag{19}$$

$$= \sum_{\mathbf{x}: \frac{p(\mathbf{x})}{q(\mathbf{x})} \geq 1, \nu(\mathbf{x}) \geq 0} q(\mathbf{x}) \left(\frac{p(\mathbf{x})}{q(\mathbf{x})} - 1\right) \nu(\mathbf{x}) + \sum_{\mathbf{x}: \frac{p(\mathbf{x})}{q(\mathbf{x})} < 1, \nu(\mathbf{x}) \geq 0} q(\mathbf{x}) \left(\frac{p(\mathbf{x})}{q(\mathbf{x})} - 1\right) \nu(\mathbf{x})$$

$$+ \sum_{\mathbf{x}: \frac{p(\mathbf{x})}{q(\mathbf{x})} \geq 1, \nu(\mathbf{x}) \leq 0} q(\mathbf{x}) \left(\frac{p(\mathbf{x})}{q(\mathbf{x})} - 1\right) \nu(\mathbf{x}) + \sum_{\mathbf{x}: \frac{p(\mathbf{x})}{q(\mathbf{x})} < 1, \nu(\mathbf{x}) \leq 0} q(\mathbf{x}) \left(\frac{p(\mathbf{x})}{q(\mathbf{x})} - 1\right) \nu(\mathbf{x}).$$

Each of the four terms above can be bounded independently as follows: for the first two terms, we multiply by $\frac{p(\mathbf{x})}{q(\mathbf{x})}$, the third term is clearly negative, and the final term is upper bounded by

multiplying by $1 + \frac{p(\mathbf{x})}{q(\mathbf{x})}$:

$$\sum_{\mathbf{x}: \frac{p(\mathbf{x})}{q(\mathbf{x})} \geq 1, \nu(\mathbf{x}) \geq 0} q(\mathbf{x}) \left( \frac{p(\mathbf{x})}{q(\mathbf{x})} - 1 \right) \nu(\mathbf{x}) \leq \sum_{\mathbf{x}: \frac{p(\mathbf{x})}{q(\mathbf{x})} \geq 1, \nu(\mathbf{x}) \geq 0} q(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} \left( \frac{p(\mathbf{x})}{q(\mathbf{x})} - 1 \right) \nu(\mathbf{x}) \tag{20}$$

$$\sum_{\mathbf{x}: \frac{p(\mathbf{x})}{q(\mathbf{x})} < 1, \nu(\mathbf{x}) \geq 0} q(\mathbf{x}) \left( \frac{p(\mathbf{x})}{q(\mathbf{x})} - 1 \right) \nu(\mathbf{x}) \leq \sum_{\mathbf{x}: \frac{p(\mathbf{x})}{q(\mathbf{x})} \leq 1, \nu(\mathbf{x}) \geq 0} q(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} \left( \frac{p(\mathbf{x})}{q(\mathbf{x})} - 1 \right) \nu(\mathbf{x}) \tag{21}$$

$$\sum_{\mathbf{x}: \frac{p(\mathbf{x})}{q(\mathbf{x})} \geq 1, \nu(\mathbf{x}) \leq 0} q(\mathbf{x}) \left( \frac{p(\mathbf{x})}{q(\mathbf{x})} - 1 \right) \nu(\mathbf{x}) \leq 0 \tag{22}$$

$$\sum_{\mathbf{x}: \frac{p(\mathbf{x})}{q(\mathbf{x})} < 1, \nu(\mathbf{x}) \leq 0} q(\mathbf{x}) \left( \frac{p(\mathbf{x})}{q(\mathbf{x})} - 1 \right) \nu(\mathbf{x}) \leq \sum_{\mathbf{x}: \frac{p(\mathbf{x})}{q(\mathbf{x})} < 1, \nu(\mathbf{x}) \leq 0} q(\mathbf{x}) \left( \frac{p(\mathbf{x})}{q(\mathbf{x})} - 1 \right) \left( 1 + \frac{p(\mathbf{x})}{q(\mathbf{x})} \right) \nu(\mathbf{x}). \tag{23}$$

Finally, for each of these terms, we can now upper bound $\nu(\mathbf{x})$ by its maximum absolute value, $|\nu(\mathbf{x})| \leq \nu_0$, and combine the terms to get the following bound on $\Delta_\nu(p, q)$:

$$\Delta_\nu(p, q) \leq \nu_0 \sum_{\mathbf{x}: \nu(\mathbf{x}) > 0} p(\mathbf{x}) \left( \frac{p(\mathbf{x})}{q(\mathbf{x})} - 1 \right) + 0 + \nu_0 \sum_{\mathbf{x}: \frac{p(\mathbf{x})}{q(\mathbf{x})} < 1, \nu(\mathbf{x}) < 0} \left( \frac{p^2(\mathbf{x})}{q(\mathbf{x})} - q(\mathbf{x}) \right) \tag{24}$$

$$\leq \nu_0 \left[ \sum_{\mathbf{x}} \frac{p^2(\mathbf{x})}{q(\mathbf{x})} - 1 \right], \tag{25}$$

where Equation 25 follows from using the fact that for the case when $p(\mathbf{x})/q(\mathbf{x}) > 1$ but $\nu(\mathbf{x}) \leq 0$, $p(\mathbf{x}) \left( \frac{p(\mathbf{x})}{q(\mathbf{x})} - 1 \right) > 0$, and hence it upper bounds the RHS of Equation 22. For the last case, where $\mathbf{x} : \frac{p(\mathbf{x})}{q(\mathbf{x})} < 1, \nu(\mathbf{x}) < 0$, we note that $\sum_{\mathbf{x}: \frac{p(\mathbf{x})}{q(\mathbf{x})} < 1, \nu(\mathbf{x}) < 0} q(\mathbf{x}) \geq \sum_{\mathbf{x}: \frac{p(\mathbf{x})}{q(\mathbf{x})} < 1, \nu(\mathbf{x}) < 0} p(\mathbf{x})$, and hence the upper bound on this term in Equation 25 follows. To complete the argument note that $D_{\text{CQL}}$ exactly takes the form obtained in the final equation, and hence:

$$\Delta_\nu(p, q) \leq \nu_0 \cdot D(p, q).$$

We can now use this result to bound the return differences, by using standard results for bounding the performance difference between policies (Achiam et al., 2017; Schulman et al., 2015) in terms of $\frac{1}{(1-\gamma)^2} \times D(\pi, \pi_\beta)$. At the core of these results is a bound on $\mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s})}[f(\mathbf{s}, \mathbf{a})] - \mathbb{E}_{\mathbf{a} \sim \pi_\beta(\cdot|\mathbf{s})}[f(\mathbf{s}, \mathbf{a})]$, and hence the result proven above directly applies. This proves the required result. □

We will now provide a proof for Theorem 3.1.

**Theorem B.1** (Theorem 3.1 restated)**.** *W.h.p. $\geq 1 - \delta$, for any prescribed level of safety $\zeta$, the maximum possible policy improvement over choices of $\alpha$, $J(\pi_\alpha) - J(\pi_\beta) \leq \zeta^+$, where $\zeta^+$ is given by:*

$$\zeta^+ := \max_\alpha \quad h^*(\alpha) \cdot \frac{1}{(1-\gamma)^2} \quad \text{s.t.} \quad \frac{c_1}{(1-\gamma)^2} \sqrt{\frac{C^{\pi_\alpha}_{diff}}{|\mathcal{D}|}} - \frac{\alpha}{1-\gamma} \mathbb{E}_{\mathbf{s} \sim \widehat{d}^{\pi_\alpha}}[D(\pi_\alpha, \pi_\beta)(\mathbf{s})] \leq \zeta, \tag{26}$$

*where $h^*$ is a monotonically decreasing function of $\alpha$, and $h(0) = \mathcal{O}(1)$.*

***Proof of Theorem B.1.*** To prove this theorem, we will apply Lemma B.1 on $x_i = \frac{D(\pi_\alpha(\cdot|\mathbf{s}_i)||\pi_\beta(\cdot|\mathbf{s}_i))}{\mu(\mathbf{s}_i)}$ and combine it with a the safe policy improvement guarantee for behavior regularization methods that admit updates of the form shown in Equation 1.

First, we note by applying Lemma B.1 in its expectation form that:

$$\left( \mathbb{E}_{\mathbf{s} \sim \widehat{d}^{\pi_\alpha}} \left[ \sqrt{\frac{D(\pi_\alpha(\cdot|\mathbf{s})||\pi_\beta(\cdot|\mathbf{s}))}{\mu(\mathbf{s})}} \right] \right)^2 \geq \mathbb{E}_{\mathbf{s}_1 \sim \widehat{d}^{\pi_\alpha}, \mathbf{s}_2 \sim \widehat{d}^{\pi_\alpha}} \left[ \sqrt{\frac{D(\pi_\alpha(\cdot|\mathbf{s}_1)||\pi_\beta(\cdot|\mathbf{s}_1))}{\mu(\mathbf{s}_1)}} - \sqrt{\frac{D(\pi_\alpha(\cdot|\mathbf{s}_2)||\pi_\beta(\cdot|\mathbf{s}_2))}{\mu(\mathbf{s}_2)}} \right]^2,$$

where the term on the RHS of the above equation corresponds to $C^{\pi}_{\text{diff}}$.

Now we can plug this into the safe-policy improvement guarantee to obtain the resulting result as follows. Note that the first term in the bound in Equation 13 can be lower bounded using the differential concentrability as discussed above, and therefore, we get the following lower bound on $\zeta$:

$$\zeta \geq \mathcal{O}\left(\frac{1}{(1-\gamma)^2}\right)\sqrt{\frac{C^{\pi_\alpha}_{\text{diff}}}{|\mathcal{D}|}} + \alpha\mathbb{E}_{\mathbf{s}\sim\widehat{d}^{\pi_\alpha}}\left[D(\pi_\alpha,\pi_\beta)(\mathbf{s})\right], \tag{27}$$

which is exactly the same as the expression for the constraint in Theorem 3.1.

Next we provide an upper bound on the maximal improvement that can be possible, in terms of $D(\pi_\alpha,\pi_\beta)$. For this, we will utilize Lemma B.2, and we can directly upper bound $J(\pi_\alpha) - J(\pi_\beta)$ as follows:

$$J(\pi_\alpha) - J(\pi_\beta) \lesssim \frac{1}{(1-\gamma)^2}\mathbb{E}_{\mathbf{s}\sim\widehat{d}^{\pi_\alpha}}\left[D(\pi_\alpha,\pi_\beta)(\mathbf{s})\right]\cdot R_{\max}. \tag{28}$$

Finally, we express this upper bound in terms of $\alpha$. Since the generic distributional constraint algorithm (Equation 1) optimizes $\mathbb{E}_{\mathbf{s}\sim\widehat{d}^{\pi}}[D(\pi,\pi_\beta)(\mathbf{s})]$ weighted by $\alpha$, we get that:

$$\mathbb{E}_{\mathbf{s}\sim\widehat{d}^{\pi_\alpha}}\left[D(\pi_\alpha,\pi_\beta)(\mathbf{s})\right] \leq h^*(\alpha), \tag{29}$$

where $h^*(\alpha)$ is a decreasing function of $\alpha$. Therefore, the maximal improvement is upper bounded by: $h^*(\alpha)\mathcal{O}\left(\frac{1}{(1-\gamma)^2}\right)$, which completes the proof of this theorem. □

### B.3 Proof of Lemma 4.1

**Lemma B.3** ((Lemma 4.1 restated) Per-state modification of Q-values.)**.** *Let $g$ represents $g\left(\tau\cdot\pi(\cdot|\mathbf{s})\right)$. The Q-function obtained after one TD-learning iteration using the objective in Eq. 10 is:*

$$Q_\theta(\mathbf{s},\mathbf{a}) := \mathcal{B}^\pi\bar{Q}(\mathbf{s},\mathbf{a}) - \alpha\frac{\pi(\mathbf{a}|\mathbf{s}) + \pi_\beta(\mathbf{a}|\mathbf{s})g - 2\pi_\beta(\mathbf{a}|\mathbf{s})}{2\pi_\beta(\mathbf{a}|\mathbf{s})} \tag{30}$$

*where $\mathcal{B}^\pi\bar{Q}(\mathbf{s},\mathbf{a})$ is the Bellman backup operator applied to a delayed target Q-network.*

Recall from Section 2 that the objective of CQL consists of two terms

$$\min_\theta\ \alpha\underbrace{\left(\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi}\left[Q_\theta(\mathbf{s},\mathbf{a})\right] - \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}\left[Q_\theta(\mathbf{s},\mathbf{a})\right]\right)}_{\mathcal{R}(\theta)} + \frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[\left(Q_\theta(\mathbf{s},\mathbf{a}) - \mathcal{B}^\pi\bar{Q}(\mathbf{s},\mathbf{a})\right)^2\right], \tag{31}$$

where $\mathcal{B}^\pi\bar{Q}(\mathbf{s},\mathbf{a})$ is the Bellman backup operator applied to a delayed target Q-network. In tabular setting, the Q-function obtained after one iteration of TD-learning using the objective function in Eq. 31 is given by:

$$Q_\theta(\mathbf{s},\mathbf{a}) := \mathcal{B}^\pi\bar{Q}(\mathbf{s},\mathbf{a}) - \alpha\left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right]. \tag{32}$$

In CQL, the result in Eq. 32 is obtained by setting the derivative of the objective in Eq. 31 with respect to the Q-values to 0, and solve for $Q_\theta(\mathbf{s},\mathbf{a})$ (Kumar et al., 2020).

We now restate the new regularizer introduced by ReDS, and the new objective function for the Q-function.

$$\mathcal{R}(\theta;\rho) = \left(\frac{1}{2}\mathop{\mathbb{E}}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi}\left[Q_\theta(\mathbf{s},\mathbf{a})\right] + \frac{1}{2}\mathop{\mathbb{E}}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\rho}\left[Q_\theta(\mathbf{s},\mathbf{a})\right] - \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}\left[Q_\theta(\mathbf{s},\mathbf{a})\right]\right) \tag{33}$$

$$\min_{\theta} \quad J_Q(\theta) = \mathcal{R}(\theta; \rho) + \frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[\left(Q_\theta(\mathbf{s},\mathbf{a}) - \mathcal{B}^\pi\bar{Q}(\mathbf{s},\mathbf{a})\right)^2\right] \tag{34}$$

Notice that the main difference between the original CQL objective in Eq. 31 and the new objective in Eq. 34 is the distribution with which we push down Q values. The objective in Eq. 31 pushes Q values down under the learned policy $\pi$, whereas the objective in Eq. 34 pushes Q values down under a mixture of $\pi$ and $\rho$, i.e. $\frac{1}{2}\pi + \frac{1}{2}\rho$. Since $\rho$ is parameterized by a neural network whose input does not contain the Q values, its gradient with respect to the Q values is 0. Additionally, since the mixture $\frac{1}{2}\pi + \frac{1}{2}\rho$ plays the same role in Eq.33 that $\pi$ plays in the objective function in Eq. 31, we therefore can obtain the solution for the Q-values after updating the Q-function using the objective function in Eq. 34 simply by replacing $\pi$ in Eq. 32 with the mixture. That is, in tabular setting, after updating the Q-function using the objective in Eq.34, the Q-values are:

$$\begin{aligned}
Q_\theta(\mathbf{s},\mathbf{a}) &= \mathcal{B}^\pi\bar{Q}(\mathbf{s},\mathbf{a}) - \alpha\left[\frac{\frac{1}{2}\pi(\mathbf{a}|\mathbf{s}) + \frac{1}{2}\rho(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right]. \\
&= \mathcal{B}^\pi\bar{Q}(\mathbf{s},\mathbf{a}) - \alpha\left[\frac{\pi(\mathbf{a}|\mathbf{s}) + \rho(\mathbf{a}|\mathbf{s})}{2\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right]. \\
&= \mathcal{B}^\pi\bar{Q}(\mathbf{s},\mathbf{a}) - \alpha\left[\frac{\pi(\mathbf{a}|\mathbf{s}) + \rho(\mathbf{a}|\mathbf{s}) - 2\pi_\beta(\mathbf{a}|\mathbf{s})}{2\pi_\beta(\mathbf{a}|\mathbf{s})}\right]. \\
&= \mathcal{B}^\pi\bar{Q}(\mathbf{s},\mathbf{a}) - \alpha\left[\frac{\pi(\mathbf{a}|\mathbf{s}) + \pi_\beta(\mathbf{a}|\mathbf{s})g(.) - 2\pi_\beta(\mathbf{a}|\mathbf{s})}{2\pi_\beta(\mathbf{a}|\mathbf{s})}\right].
\end{aligned}$$

since $\rho$ subsumes $\pi_\beta \cdot g$, giving us the desired result.

### B.4 Proofs of Lemma 4.2

To prove Lemma 4.2, we will consider the following abstract update form for the policy evaluation version of CQL (ReDS), that obtains the next Q-function iterate $Q_{k+1}$:

$$\min_Q \alpha\left(\mathop{\mathbb{E}}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi^{re}}[Q(\mathbf{s},\mathbf{a})] - \mathop{\mathbb{E}}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi_\beta}[Q(\mathbf{s},\mathbf{a})]\right) + \frac{1}{2}\mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[\left(Q(\mathbf{s},\mathbf{a}) - \mathcal{B}^\pi Q_k(\mathbf{s},\mathbf{a})\right)^2\right], \tag{35}$$

**Lemma B.4** (CQL (ReDS) restated more completely.)**.** *CQL (ReDS) solves the following optimization problem, when $\alpha$ is large enough:*

$$\max_\pi \ \widehat{J}(\pi) - \frac{\alpha}{2(1-\gamma)}\mathbb{E}_{\mathbf{s}\sim\widehat{d}^\pi}\left[D(\pi,\pi_\beta)(\mathbf{s}) + \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}\left[g\left(\frac{1}{\tau\cdot\pi(\mathbf{a}|\mathbf{s})}\right)\mathbb{I}\left\{\pi_\beta(\mathbf{a}|\mathbf{s}) \geq \varepsilon\right\}\right]\right].$$

*Proof.* For proving Lemma 4.2, we follow an argument similar to the proof of Theorem 3.1 from Kumar et al. (2020). By differentiating the above objective w.r.t. Q, we note that the ReDS + CQL objective above exhibits the following effective Bellman backup

$$\forall \mathbf{s},\mathbf{a} \in \mathcal{D}, Q_{k+1}(\mathbf{s},\mathbf{a}) := \left(\mathcal{B}^\pi Q_k\right)(\mathbf{s},\mathbf{a}) - \alpha\left(\frac{\pi^{re}(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right). \tag{36}$$

This backup is equivalent to running pessimistic RL with a reward bonus equal to $-\alpha\left(\frac{\pi^{re}(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right)$, and therefore, the policy obtained by maximizing the resulting Q-function can be expressed as:

$$\max_\pi \ \widehat{J}(\pi) - \alpha\frac{1}{1-\gamma}\mathbb{E}_{\mathbf{s}\sim\widehat{d}^\pi}\left[\mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}\left[\left(\frac{\pi^{re}(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right)\right]\right] \tag{37}$$

$$\equiv \max_\pi \ \widehat{J}(\pi) - \alpha\frac{1}{2(1-\gamma)}\mathbb{E}_{\mathbf{s}\sim\widehat{d}^\pi}[D(\pi,\pi_\beta)(\mathbf{s})] - \frac{\alpha}{2(1-\gamma)}\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\widehat{d}^\pi}\left[\mathbb{I}\{\pi_\beta(\mathbf{a}|\mathbf{s}) > 0\}g\left(\frac{1}{\tau\cdot\pi(\mathbf{a}|\mathbf{s})}\right)\right]. \tag{38}$$

To finish the proof, we redefine the notion of support $\mathbb{I}\{\pi_\beta(\mathbf{a}|\mathbf{s}) > 0\}$ using $\mathbb{I}\{\pi_\beta(\mathbf{a}|\mathbf{s}) > \varepsilon\}$ for a small-enough $\varepsilon$, and this gives us the desired form. To see why we can do this, note that for any policy $\pi$ such that $\pi(\mathbf{a}|\mathbf{s}) > \delta$, when $\pi_\beta(\mathbf{a}|\mathbf{s}) < \varepsilon$, $D(\pi, \pi_\beta)$ can be made to increase arbitrarily since $\pi_\beta(\mathbf{a}|\mathbf{s})$ appears in the denominator, and this would make the objective value in Equation 37 for such a $\pi$ correspondingly small, and hence such a $\pi$ would not be optimal when $\alpha$ is chosen to be sufficiently large enough. Therefore, we can discard such a policy, $\pi$ and restrict the term to use $\pi_\beta(\mathbf{a}|\mathbf{s}) > \varepsilon$ instead, as the optimal solution to Equation 37 would then satisfy: $\pi(\mathbf{a}|\mathbf{s}) > 0 \implies \pi_\beta(\mathbf{a}|\mathbf{s}) \geq \varepsilon$. □

# C Implementation details of CQL (ReDS)

In this section, we will provide implementation details about our algorithm, CQL (ReDS). The pseudo-code in Algorithm 1 illustrates the different update steps of our algorithms. In addition, we provided a detailed python-like algorithm description for ease of implementation. This can be found below in Section C.1.

Most of the components of Algorithm 1 are straightforward and follow the same convention, training update and, as we will discuss, hyperparameters as the CQL algorithm. This includes training the policy $\pi_\phi$, and for the most part training the critic $Q_\theta$. The main difference in the update for CQL (ReDS) is utilizing the mixture of $\pi$ and $\rho$ in the CQL regularizer. For obtaining $\rho_\psi$, we utilize a standard advantage-weighted training update, following the papers (Kostrikov et al., 2021a; Nair et al., 2020b; Peng et al., 2019). Following these prior works, we also clip the argument to the exponent between a minimum range and a maximum range to be numerically stable:

$$\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[\log \rho_\psi(\mathbf{a}|\mathbf{s}) \cdot \exp\left[\text{clip}\left(-A^\pi_\theta(\mathbf{s},\mathbf{a})/\tau, \sigma_{\min}, \sigma_{\max}\right)\right]. \tag{39}$$

In our experiments, we chose $\sigma_{\min} = -10$ and $\sigma_{\max} = 5$ across all the tasks and domains we study. These details are standard in training advantage-weighted algorithms.

## C.1 Detailed algorithm description for CQL (ReDS)

Algorithm 1 provides the pseudo-code for CQL (ReDS). We provide the detailed description of how each update step in Algorithm 1 is implemented using Python syntax based on the PyTorch Framework in this section. We include 3 code listings below, illustrating the update steps for the parametric Q-functions, the policy and the learnt distribution $\rho$.

Listing 1: Training Q networks given a batch of data, corresponding to step 3 in Algorithm 1

```
q_data = critic(batch['observations'], batch['actions'])

next_dist = actor(batch['next_observations'])
next_pi_actions, next_log_pis = next_dist.sample()

target_qval = target_critic(batch['observations'],
                            next_pi_actions)
target_qval = batch['rewards'] + \
    self.gamma * (1 - batch['dones']) * target_qval

td_loss = mse_loss(q_data, target_qval)

# importance sampling term
num_samples = 4

# assume env is normalized between [-1, 1]
random_actions = uniform_sample((num_samples,
    batch['actions'].shape[-1]), min=-1, max=1)
random_pi = 0.5 ** batch['actions'].shape[-1]

dist = actor(batch['observations'])
pi_actions, log_pis = dist.sample(num_samples)

rho_dist = rho(batch['observations'])
rho_actions, log_probs_rho = rho_dist.sample(num_samples)

q_rand_is = critic(batch['observations'],
                   random_actions) - random_pi
q_pi_is = critic(batch['observations'],
                 pi_actions) - log_pis
q_rho_is = critic(batch['observations'],
                  rho_actions) - log_probs_rho

cat_q = concatenate(q_rand_is, q_pi_is, new_axis=True)
cat_q = logsumexp(cat_q, axis=-1)

cat_q_rho = logsumexp(q_rho_is, axis=-1)

# average between rho and pi
push_down_term_reds = 0.5 * (cat_q + cat_q_rho)

reds_loss = td_loss + \\
    ((push_down_term_reds - q_data).mean() * cql_alpha)

critic_optimizer.zero_grad()
reds_loss.backward()
critic_optimizer.step()
```

Listing 2: Training the policy (or the actor) given a batch of data (step 4 in Algorithm 1)

```
# Identical to CQL
# return distribution of actions
dist = actor(batch['observations'])

# sample actions with associated log probabilities
pi_actions, log_pis = dist.sample()

# calculate q value of actor actions
qpi = critic(batch['observations'], actions)
qpi = qpi.min(axis=0)

# same objective as CQL (kumar et al.)
actor_loss = (log_pis * self.alpha - qpi).mean()

# optimize loss
actor_optimizer.zero_grad()
actor_loss.backward()
actor_optimizer.step()
```

Listing 3: Training the $\rho_\psi$ distribution given a batch of data (step 5 in Algorithm 1)

```
# AWR style update to find rho

# sample policy actions for advantage calculation
dist = actor(batch['observations'])
pi_actions, log_pis = dist.sample()

# calculate advantage
qdata = critic(batch['observations'], batch['actions'])
value = critic(batch['observations'], actions)
advantage = (qdata - value.min(0)).mean()

# awr style clipping
clipped_advantage = clip(advantage/self.temperature, \
    min=-10, max=5)

# find log rho(a|s)
rho_dist = _rho(batch['observations'])
log_prob_rho = rho_dist.log_prob(batch['actions'])

# Advantage Weighted Log Probabilities is the loss for rho
rho_loss = -(exp(-clipped_advantage) * log_prob_rho)
rho_loss = rho_loss.mean()

rho_optimizer.zero_grad()
rho_loss.backward()
rho_optimizer.step()
```

## D TASK AND DATASET DESCRIPTIONS

In this section, we will describe the various tasks we introduce in this paper. We also provide qualitative descriptions of these tasks here.

**Heteroskedastic antmaze navigation.** We introduce four new antmaze datasets which exhibit two different dataset distributions each for the medium and large mazes from D4RL (Fu et al., 2020). We reuse the layouts of the mazes directly from D4RL, but modify the data collection protocol. For the `noisy` datasets, given an observation from the environment, we first compute the action that would have been taken by the D4RL behavior policy, and then add Gaussian noise to the action. While this alone is not much harder, crucially, note that the variance of this added Gaussian noise differs depending on the location of the Ant in the 2D Maze. In addition there is a small bias added to the D4RL behavior policy, but this bias is dominated by noise. We present the noise standard deviations (indicated "Noise") and the bias added (indicated "Bias") for this dataset as a function of different location intervals in the maze in the left part of the Figure 8 below.

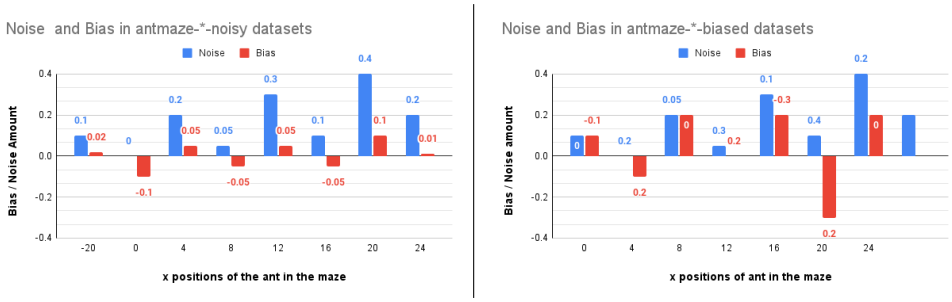


Figure 8: The distribution of noise and bias in the heteroskedastic antmaze datasets as a function of the x-position of the ant in the maze. While the noisy datasets primarily add noise, the biased datasets also add significant bias beyond the noise. The value of a given bar is the variance of the noise / bias added in the region between the x-position for that bar, and the next one.

For the `biased` datasets, in addition to adding location-dependent Gaussian noise to the action computed by D4RL behavior policies, we add a strong bias to the action (see Figure 8 (right)). Crucially note that the direction of this bias (i.e., the sign) changes based on the location of the Ant in the 2D maze, which mimics the scenario studied in our didactic navigation example in Section 3.1. In summary, because in some 2D regions of the maze, the values of the noise and bias added to the actions are larger, while in other 2D regions, they are smaller, the new offline datasets contain more heteroskedastic data distribution, where an optimal learned policy must deviate away from the data distribution much more in certain regions, whereas much lesser in other regions, which would correspond to an increase in the differential concentrability. This is demonstrated quantitatively in Table 4a. Thus, we expect that learning well on these tasks modulating the strength of the distributional constraints per state.

**Visual robotic pick and place.** We introduce a pick and place dataset, which exhibits a unique dataset distribution for a robotic pick and place manipulation task, building on the framework from Singh et al. (2020). As shown in Figure 10, the robotic setup is a 6-DOF WidowX robot in front of a green bowl with 2 objects: a target object (the ball in this case) and a distractor object (the can). The objective is to place the target object into the bin. The reward function is a sparse, binary indicator of success, where a +1 reward is given when the object is placed in the bin. This task must be done from $128 \times 128 \times 3$ raw visual observations, without access to either the robot state, or the state of the objects, which can change as the objects can roll on the surface.

**Visual robotic bin sorting.** We introduce a bin sorting tasks, which are also built on the framework from Singh et al. (2020). As shown in Figure 10, the robotic setup is a 6-DOF WidowX robot in front of two identical white bins with 2 objects to sort. The objective is to sort each object into its respective bins the target object in to the bin. The reward function is a sparse, binary indicator of success, where a +1 reward is given when both objects are placed in their correct bins. This task must be done from $128 \times 128 \times 3$ raw visual observations, without access to either the robot state, or the state of the objects, which can change as the objects can roll on the surface.

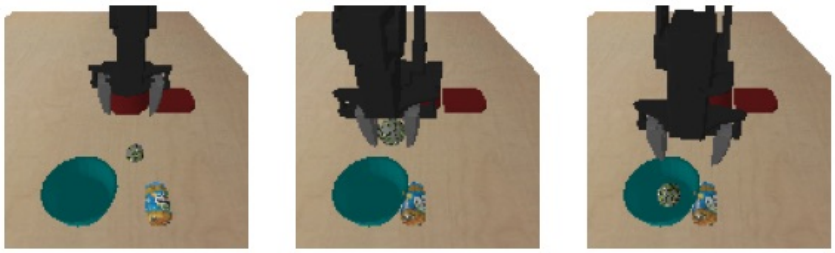

Figure 9: **Visualizing a sample trajectory for the visual pick-place robotic manipulation task.** Here is an example successful trajectory in the dataset collected using a scripted policy. Th robot reaches for the target object (the green ball), lifts it, and places it inside of the green container.

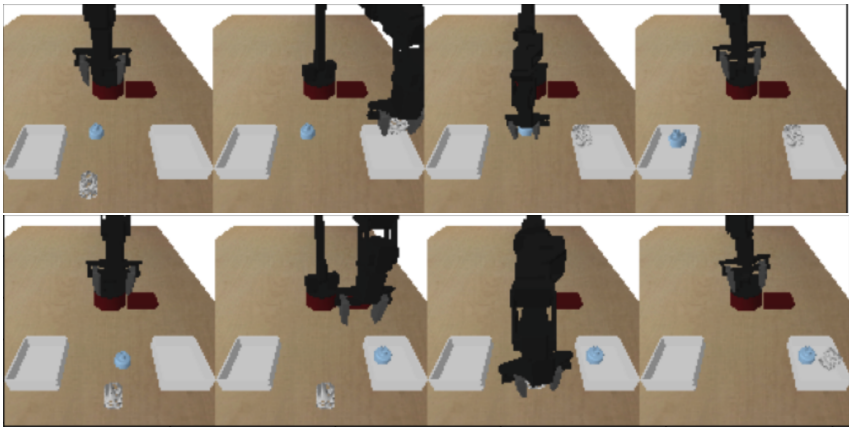

Figure 10: **Visualizing sample trajectories for the visual bin sorting robotic manipulation task.** Here are two sample trajectories for the binsorting domain. **Top:** A successful trajectory in the dataset. Here the robot places the white cylinder in the right bin and the blue ball in the left bin, succesfully sorting the objects into their respective bins. **Bottom:** A unsuccessful trajectory in the dataset. Here the robot places both objects in the same bin. This is unsuccessful as an object was placed in the incorrect bin thereby not sorting them in a correct manner.

To collect a heteroskedastic dataset, we run data collection using hardcoded scripted control policies, whose success rate and variance can be controlled. Each trajectory in the dataset was collected in the following manner. For the first phase, the robot reaches towards the object without any bias and grasps it with a reasonable success rate. Here, though, the noise and stochasticity in the scripted data collection and the inaccuracies in the scripted policy make it not succeed for every trial. During the second phase, where the robot places the object in the bin, there was a bias towards placing the object in a position in the workspace that does not correspond to the target location of the bin for which the robot can attain a reward, and which the robot will observe during evaluation. For our experiments, this bias was 85%. This forces the data distribution to be heteroskedastic: for the picking segment of this task, the behavior policy is centered around the desired optimal behavior i.e., grasping the object successfully, whereas for the placing segment, the behavior is biased towards carrying the object to the incorrect regions, requiring significant deviations from the behavior policy to succeed. An algorithm is now required to have a non-uniform amount of closeness to the behavior policy.

For the bin sorting domain, the easy domain had no bias and instead the scripted policy had additive gaussian white noise with a fixed variance which leads to low heteroskedacity.

**Atari game playing.** For the Atari tasks we consider in the paper, we devised a heteroskedastic data composition based on the DQN Replay dataset (Agarwal et al., 2020) which comprises of transitions found in the replay buffer of a run of an online DQN. Since this dataset consist of all the policies that the DQN agent produced over the course of training, and since Atari typically uses $\epsilon$-greedy exploration, where the value of $\epsilon$ decays over time, different trajectories of this dataset are generated

from different behavior policies, that all have different levels stochasticity. Naturally, since the value of $\epsilon$ decays over training and the performance of online DQN increases, the trajectories with higher return generally correlate with having lower stochasticity.

Given this information, we attempted to subsample a dataset that is heteroskedastic. For this purpose, we first divide the full replay buffer into $N$ equal chunks, where chunk 0 consists to experience observed earliest in training, while the chunk $N-1$ consists of experience seen near the final parts of training. Then, we subsample 20% of the trajectories from each of these chunks independently to obtain an intermediate dataset that comes from multiplies policies, observed at different times while training online DQN. Then, for any given trajectory $\tau$ of length $L$ in the replay chunk $i$, we only retain the transitions occurring between time steps $\lfloor \frac{(L-i)\times N}{L} \rfloor : \lfloor \frac{(L-i+1)\times N}{L} \rfloor$ in our final dataset and discard all the remaining transitions. This essentially means that the data closer to the initial states of the game comes from a good, less stochastic policy, whereas the data close to the final states of the game from a worse, highly stochastic policy. We develop two such datasets corresponding to $N = 2$ and $N = 5$ chunks. These chunks concatenated together construct the replay buffer of transitions that correspond to the 2 and 5 policy experiments seen in Section 5.

To see why this data is heteroskedastic, note that at different states of the game, we observe actions with different amounts of stochasticity and bias. This is because, as the game progresses, the effective behavior policy induced by the offline dataset exhibits a bias towards suboptimal actions (from the chunks that are earlier in DQN training) while also exhibiting substantial noise. The states that are closer to the initial states of the game, on the other hand, have an effective behavior policy that is primarily centered around a good action, with very little noise. In order to succeed, an offline RL algorithm must have different amount of conservatism at different states.

In our experiments, we considered 10 games including several standard games, and this is a subset of games studied in prior work (Kumar et al., 2021). The games we considered are: ASTERIX, BREAKOUT, Q*BERT, SEAQUEST, SPACEINVADERS, BEAMRIDER, MSPACMAN, WIZARDOFWOR, JAMESBOND, PONG.

# E EXPERIMENTAL DETAILS

For our experiments on the AntMaze domains, we built on the following open-source implementation of CQL: https://github.com/young-geng/JaxCQL, for our visual robotic experiments, we utilized our own port of the following implementation from Singh et al. (2020) in Jax: https://github.com/avisingh599/cog, and for our Atari experiments, we use the official implementation of CQL built on Dopamine (Castro et al., 2018): https://github.com/aviralkumar2907/CQL/tree/master/atari. For certain baselines (e.g EDAC, BEAR), we utilize the source implementation to stay consistent with the author's tested and tuned implementation. We additionally verified the results for D4RL benchmark for these tasks. We will summarize the hyperparameters in the next sections.

## E.1 HYPERPARAMETERS FOR CQL (REDS)

**Antmaze domains.** For the AntMaze domains, we utilized a temperature parameter $\tau = 0.3$ in our experiments (found by sweeping over $\tau \in [0.1, 0.3, 1.0, 5.0]$), for all the four dataset types in Table 4b. Every other hyperparameter was kept identical to CQL, which for the case of antmaze corresponds to applying the CQL regularizer $\mathcal{R}(\theta)$ with the dual version of CQL, where the threshold on the CQL regularizer is specified to be 1.0. Following CQL, we used 3-hidden layer critic and actor networks with layers of size 256, a critic learning rate of 3e-4 and an actor learning rate of 1e-4. We utilized the Bellman backup that computes the target value by performing a maximization over target values computed for $k = 10$ actions sampled from the policy at the next state.

**Atari domains.** For our Atari experiments, we tuned the value of $\alpha$ in CQL (Equation 2) between two values $[0.1, 0.2]$, and present the sensitivity results in Figure 4, and found that $\alpha = 0.1$ work better for CQL. We swept the value of $\tau \in [2.0, 5.0, 7.0]$ and report the sensitivity sweep in Figure 11.

**Visual Robotic Domains.** For the visual pick and place domains, we follow exactly the same hyperparameters as the CQL implementation from COG (Singh et al., 2020): a critic learning rate

of 3e-4, an actor learning rate of 1e-4, using $k = 4$ actions from the policy for computing the target values for computing the TD error, and using $k = 4$ actions to compute the log-sum-exp in CQL. For the value of $\tau$, we swept over $\tau \in [0.1, 1.0, 10.0, 100.0]$, and used a $\tau = 1.0$ for our experiments.

## F Additional Ablation Studies

In this section, we present some results of an ablation study of the performance of CQL (ReDS) with respect to the temperature hyperparameter $\tau$ that appears in Equation 9. Before discussing the results, let us intuitively aim to understand the significance of this hyperparameter. When $\tau$ is extremely small we would expect $\rho_\psi$ to be a distribution centered at the worst possible action, within the support of the behavior policy. When $\tau$ is large, we would expect the learned $\rho_\psi$ to be close to the behavior policy, since the exponentiated advantage term would essentially behave as a constraint against a uniform distribution. Neither of these extremes are desirable, while the former does not behave much differently than a distributional constraint (except that the Q-value at the action with the smallest Q-value in the dataset support is not pushed up anymore), the latter also behaves like a distributional constraint, but with just half the effective multiplier $\alpha$ on the CQL regularizer. We would therefore expect an intermediate $\tau$ to perform the best.

To verify these insights, we study the sensitivity of the performance of CQL (ReDS) with respect to $\alpha$ on the Atari datasets. Our results shown in Figure 12 confirm that indeed an intermediate value of $\tau = 5.0$ out of the tested values, $\tau \in [2.0, 5.0, 7.0]$ works the best.

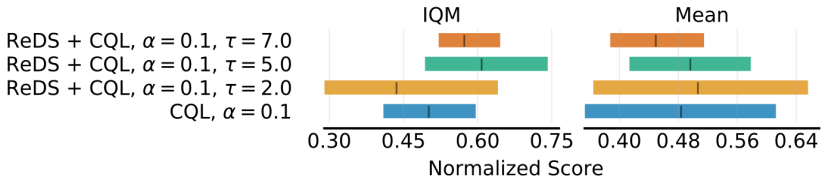


Figure 11: **Sensitivity of CQL (ReDS) to the temperature hyperparameter $\tau$ in Equation 9 evaluated on the Atari game experiments with 5 policies.** Observe that an intermediate value of temperature $\tau = 5.0$ works best

In addition, we study the sensitivity of the performance of CQL (ReDS) with respect to $\alpha$ on the Atari datasets. We report the performance for two different values of $\alpha \in \{0.1, 0.2\}$ from CQL (Equation 2) in Figure 12. Observe that CQL (ReDS) with a given $\alpha$ outperforms base CQL for the corresponding $\alpha$. Additionally note that the degradation in performance of CQL (ReDS) as $\alpha$ increases is lesser than base CQL.

## G Additional Baseline Comparison for Heteroskedastic Antmaze Navigation

In this section we will provide additional baseline comparison for REDS with two additional Offline RL methods: EDAC (An et al., 2021) and BEAR (Kumar et al., 2019).

### G.1 Hyperparameters for EDAC

As done in An et al. (2021), we tune the method over two hyperparameters. The first hyperparameter is the ensemble size $N$ which specifies the number of Q functions. The second parameter we consider is $\eta$, the weight of the ensemble gradient diversity term. Below in table 4, we show the values considered for each hyperparameter. There is significant overlap to these parameters with the ones used in the Mujoco Gym and Adroit Domains that the authors used. We utilized the publicly available code (https://github.com/snu-mllab/EDAC) released by the authors of EDAC and were able to replicate the results they reported for the D4RL MuJoCo Gym environments in An et al. (2021).

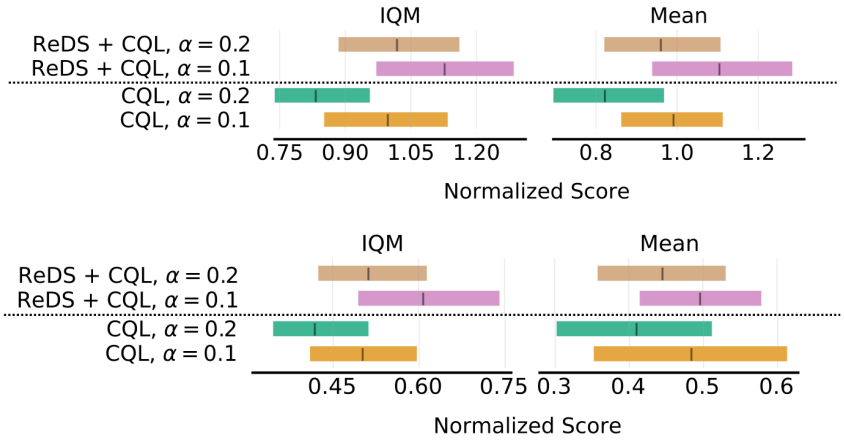


Figure 12: **Sensitivity of ReDS + CQL to the temperature hyperparameter $\alpha$ in Equation 2** We report the performance of CQL (ReDS) vs CQL on the IQM normalized score and the mean normalized score over ten Atari games, for the case of **two** (top) and **five** (bottom) policies. We consider this performance for two different values of $\alpha \in \{0.1, 0.2\}$ in CQL (Equation 2). Observe that CQL (ReDS) with a given $\alpha$ outperforms base CQL for the corresponding $\alpha$. Additionally note that the degradation in performance of ReDS (CQL) as $\alpha$ increases is lesser than base CQL.

Table 4: EDAC Hyperparameters

| **Hyperparameters** | **Values** |
|---|---|
| $N$ | 10, 20, 50, 100 |
| $\eta$ | 0, 1, 5, 10, 50, 100, 1000 |

### G.2 Hyperparameters for BEAR

As done in Kumar et al. (2019), we tuned this method over two hyperparameters. The first is the Kernel Type of the MMD between the behavior policy $\pi_\beta$ and the actor $\pi$, and found that Laplacian performed better. The second parameter considered is $\sigma$, which is needed for the Laplacian kernel as defined. Below in table 5, we show the values considered for each hyperparameter. There is significant overlap to these parameters with the ones used in the Mujoco Gym and Adroit Domains that the authors used. We utilized the publicly available code (`https://github.com/rail-berkeley/d4rl_evaluations`) released by the authors of BEAR and were able to replicate the results they reported for the D4RL MuJoCo Gym environments in Kumar et al. (2019).

Table 5: BEAR Hyperparameters

| **Hyperparameters** | **Values** |
|---|---|
| Kernel Type | Laplacian, Gaussian |
| $\sigma$ | 1, 10, 20, 50 |