Offline-to-Online Reinforcement Learning via Balanced Replay and Pessimistic Q-Ensemble

Anonymous Author(s)
Affiliation
Address
email

Abstract: Recent advance in deep offline reinforcement learning (RL) has made it possible to train strong robotic agents from offline datasets. However, depending on the quality of the trained agents and the application being considered, it is often desirable to fine-tune such agents via further online interactions. In this paper, we observe that state-action distribution shift may lead to severe bootstrap error during fine-tuning, which destroys the good initial policy obtained via offline RL. To address this issue, we first propose a balanced replay scheme that prioritizes samples encountered online while also encouraging the use of near-on-policy samples from the offline dataset. Furthermore, we leverage multiple Q-functions trained pessimistically offline, thereby preventing overoptimism concerning unfamiliar actions at novel states during the initial training phase. We show that the proposed method improves sample-efficiency and final performance of the fine-tuned robotic agents on various locomotion and manipulation tasks.

Keywords: Deep Reinforcement Learning, Offline RL, Fine-tuning

1 Introduction

Deep offline reinforcement learning (RL) [1] has the potential to train strong robotic agents without any further environment interaction by leveraging deep neural networks and huge offline datasets. Accordingly, the research community has demonstrated that offline RL can train both simulated [2, 3, 4, 5, 6, 7, 8, 9] and real [5, 10] robots that are often more performant than the behavior policy that generated the offline dataset. However, thusly trained offline RL agents may be suboptimal, for (a) the dataset they were trained on may be suboptimal; and (b) environment in which they are deployed may be different from the environment in which the dataset was generated. This necessitates an online fine-tuning procedure, where the robot improves by gathering additional samples.

Off-policy RL algorithms are well-suited for offline-to-online RL, since they can leverage both offline and online samples. Fine-tuning an offline RL agent using a conventional off-policy RL algorithm, however, is difficult due to distribution shift, i.e., the robot may encounter unfamiliar state-action regime that is not covered by the offline dataset. The Q-function cannot provide accurate value estimates for such out-of-distribution (OOD) online samples, and updates with such samples lead to severe bootstrap error. This leads to policy updates in an arbitrary direction, destroying the good initial policy obtained by offline RL.

To address state-action distribution shift, we first introduce a balanced replay scheme that enables us to provide the robotic agent with near-on-policy samples from the offline dataset, in addition to samples gathered online. Specifically, we train a network that measures the *online-ness* of available samples, then prioritize samples according to this measure. This adjusts the sampling distribution for Q-learning to be closer to online samples, which enables timely value propagation and more accurate policy evaluation in the novel state-action regime.

However, we find that the above sampling scheme is not enough, for the Q-function may be overoptimistic about unseen actions at novel online states. This misleads the robot to prefer potentially bad actions, and in turn, more severe distribution shift and bootstrap error. We therefore propose a pessimistic Q-ensemble scheme. In particular, we first observe that a specific class of offline RL

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

algorithms that train pessimistic Q-functions [8, 9] make an excellent starting point for offline-to-online RL. When trained as such, the Q-function implicitly constrains the policy to stay near the behavior policy during the initial fine-tuning phase. Building on this observation, we leverage multiple pessimistic Q-functions, which guides the robotic agent with a more high-resolution pessimism and stabilizes fine-tuning.

In our experiments, we demonstrate the strength of our method based on (1) MuJoCo [11] locomotion tasks from the D4RL [12] benchmark suite, and (2) vision-based robotic manipulation tasks from Singh et al. [10]. We show that our method achieves stable training during fine-tuning, while outperforming all baseline methods considered, both in terms of final performance and sample-efficiency. We provide a thorough analysis of each component of our method.

2 Background

Reinforcement learning. We consider the standard RL framework, where an agent interacts with the environment so as to maximize the expected total return. More formally, at each timestep t, the agent observes a state s_t, and performs an action a_t according to its policy π. The environment rewards the agent with r_t, then transitions to the next state s_{t+1}. The agent's objective is to maximize the expected return $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $\gamma \in [0, 1)$ is the discount factor. The unnormalized stationary state-action distribution under π is defined as $d^\pi(s, a) := \sum_{t=0}^{\infty} \gamma^t d_t^\pi(s, a)$, where $d_t^\pi(s, a)$ denotes the state-action distribution at timestep t of the Markov chain defined by the fixed policy π.

Soft actor-critic. We mainly consider off-policy RL algorithms, a class of algorithms that can, in principle, train an agent with samples generated by any behavior policy. In particular, soft actor-critic [SAC; 13] is an off-policy actor-critic algorithm that learns a soft Q-function $Q_\theta(s, a)$ parameterized by θ and a stochastic policy π_ϕ modeled as a Gaussian, parameterized by ϕ. SAC alternates between critic and actor updates by minimizing the following objectives, respectively:

$$\mathcal{L}^{\texttt{SAC}}_{\texttt{critic}}(\theta) = \mathop{\mathbb{E}}_{(s,a,s')\sim\mathcal{B}} \Big[\big(Q_\theta(s,a) - r(s,a) - \gamma \mathop{\mathbb{E}}_{a'\sim\pi_\phi} \big[Q_{\bar{\theta}}(s',a') - \alpha \log \pi_\phi(a'|s') \big] \big)^2 \Big], \tag{1}$$

$$\mathcal{L}^{\texttt{SAC}}_{\texttt{actor}}(\phi) = \mathop{\mathbb{E}}_{s\sim\mathcal{B}, a\sim\pi_\phi} \big[\alpha \log \pi_\phi(a|s) - Q_\theta(s,a) \big], \tag{2}$$

where $\mathcal{B}$ is the replay buffer, $\bar{\theta}$ the delayed parameters, and α the temperature parameter.

Conservative Q-learning. Offline RL algorithms are off-policy RL algorithms that utilize static datasets for training an agent. In particular, conservative Q-learning [CQL; 9] pessimistically evaluates the current policy, and learns a lower bound (in expectation) of the ground-truth Q-function. To be specific, policy evaluation step of CQL minimizes the following:

$$\mathcal{L}^{\texttt{CQL}}_{\texttt{critic}}(\theta) = \frac{1}{2} \mathop{\mathbb{E}}_{(s,a,s')\sim\mathcal{B}} \big[(Q_\theta - \mathcal{B}^{\pi_\phi} Q_{\bar{\theta}})^2 \big] + \alpha_0 \mathop{\mathbb{E}}_{s\sim\mathcal{B}} \big[\log \sum_a \exp Q(s,a) - \mathop{\mathbb{E}}_{a\sim\hat{\pi}_\beta} [Q(s,a)] \big], \tag{3}$$

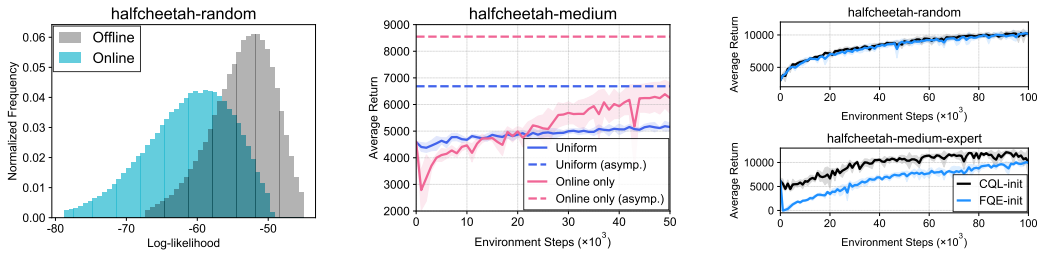
where $\hat{\pi}_\beta(a_0|s_0) := \frac{\sum_{s,a\in\mathcal{B}} \mathbb{1}[s=s_0, a=a_0]}{\sum_{s\in\mathcal{B}} \mathbb{1}[s=s_0]}$ is the empirical behavior policy, α_0 the trade-off factor, and $\mathcal{B}^\pi$ the bellman operator. The first term is the usual Bellman backup, and the second term is the regularization term that decreases the Q-values for unseen actions, while increasing the Q-values for seen actions. We argue that thusly trained pessimistic Q-function is beneficial for fine-tuning as well (see Figure 1c). Policy improvement step is the same as SAC defined in (2).

3 Fine-tuning Offline RL Agent

In this section, we investigate the distribution shift problem in offline-to-online RL. We first explain why an agent being fine-tuned can be susceptible to distribution shift, and why distribution shift is problematic. Then, we demonstrate two important design choices that decide the effect of distribution shift on fine-tuning: *sample selection* and *choice of offline Q-function*.

3.1 Distribution Shift in Offline-to-Online RL

In offline-to-online RL, there exists a distribution shift between $d^{\texttt{on}}(s, a)$ and $d^{\texttt{off}}(s, a)$, where the former denotes the state-action distribution of online samples in the online buffer $\mathcal{B}^{\texttt{on}}$, and



(a) State-action distribution shift (b) Sample selection (c) Choice of offline Q-function

Figure 1: (a) Log-likelihood estimates of (i) offline samples and (ii) online samples gathered by the offline RL agent, based on a VAE model trained on the offline dataset. (b) Fine-tuning performance on `halfcheetah-medium` task when using online samples exclusively (**Online only**), or when using both offline and online data drawn uniformly at random (**Uniform**). (c) Fine-tuning performance on `halfcheetah-random` and `halfcheetah-medium-expert` tasks, when using a pessimistic (denoted **CQL-init**) and a non-pessimistic (denoted **FQE-init**) Q-function, respectively.

the latter that of offline samples in the offline buffer $\mathcal{B}^{\texttt{off}}$. Figure 1a visualizes such distribution shift. Specifically, we trained a variational autoencoder [14] to reconstruct state-action pairs in the `halfcheetah-random` dataset that contains uniform random policy rollouts in the halfcheetah environment. Then, we compared the log-likelihood of (a) offline samples and (b) online samples collected by a CQL agent trained on the same dataset. There is a clear difference between offline and online state-action distributions.

Such distribution shift is problematic, for the agent will enter the unseen state-action regime, where Q-values (hence value estimates used for bootstrapping) can be very inaccurate. Updates in such unseen regime results in erroneous policy evaluation and arbitrary policy updates, which destroys the good initial policy obtained via offline RL. Distribution shift can be especially severe in offline-to-online RL, for the offline RL agent is often much more performant than the behavior policy (e.g., CQL can train a medium-level agent capable of running, using transitions generated by a random policy only). Also, when the offline dataset is narrowly distributed, e.g., when it is generated by a single policy, the agent is more prone to distribution shift, for the agent easily deviates from the narrow, seen distribution.

3.2 Sample Selection

In light of the above discussion, we study how sample selection affects fine-tuning. We find that online samples, which are essential for fine-tuning, are also potentially dangerous OOD samples due to distribution shift. Meanwhile, offline samples are in-distribution and safe, but leads to slow fine-tuning. As a concept experiment, we trained an agent offline via CQL (3, 2) on the `halfcheetah-medium` dataset containing medium-level transitions, then fine-tuned the agent via SAC (1, 2). We see that using online samples exclusively for updates (denoted **Online only** in Figure 1b) leads to unstable fine-tuning, where the average return drops from about 4500 to below 3000. This demonstrates the harmful effect of distribution shift, where novel, OOD samples collected online cause severe bootstrap error.

On the other hand, when using a single replay buffer for both offline and online samples then sampling uniformly at random (denoted **Uniform** in Figure 1b), the agent does not use enough online samples for updates, especially when the offline dataset is large. As a result, value propagation is slow, and as seen in Figure 1b, this scheme achieves initial stability at the cost of asymptotic performance. This motivates a balanced replay scheme that modulates the trade-off between using online samples (useful, but potentially dangerous), and offline samples (stable, but slow fine-tuning).

3.3 Choice of Offline Q-function

Another important design choice in offline-to-online RL is the offline training of Q-function. In particular, we show that a pessimistically trained Q-function mitigates the effect of distribution shift, by staying conservative about OOD actions in the initial training phase. As a concept experiment, we

compared the fine-tuning performance when using a pessimistically trained Q-function and when using a Q-function trained without any pessimistic regularization. Specifically, for a given offline dataset, we first trained a policy π_ϕ and its pessimistic Q-function $Q_{\theta_{\texttt{CQL}}}$ via CQL (3). Then we trained a non-pessimistic Q-function $Q_{\theta_{\texttt{FQE}}}$ of the pre-trained offline policy π_ϕ via Fitted Q Evaluation [FQE; 15], an off-policy policy evaluation method that trains a given policy's Q-function. Finally, we fine-tuned $\{\pi_\phi, Q_{\theta_{\texttt{CQL}}}\}$ and $\{\pi_\phi, Q_{\theta_{\texttt{FQE}}}\}$ via SAC (1, 2). See Section C for more details.

As shown in Figure 1c, both pessimistic and non-pessimistic Q-functions show similar fine-tuning performance on the `random` dataset, which contains random policy rollouts with good action space coverage. However, when fine-tuning an offline RL agent trained on the `medium-expert` dataset, which contains transitions obtained by a mixture of more selective and performant policies, non-pessimistic Q-function loses the good initial policy, reaching zero average return at one point.

The reason is that $Q_{\texttt{FQE}}$ can be overly optimistic about OOD actions at novel states when bootstrapping from them. In turn, the policy may prefer potentially bad actions, straying further away from the safe, seen trajectory. On the other hand, $Q_{\texttt{CQL}}$ remains pessimistic in the states encountered online initially, for (1) these states are incrementally different from seen states, and (2) Q-function will thus have similar pessimistic estimates due to generalization. This points to a strategy where we first train a pessimistic Q-function offline, then let it gradually lose the pessimism as the agent gains access to a balanced mix of offline and online samples via balanced replay during fine-tuning. Furthermore, since a single agent's Q-function may not be pessimistic enough, we may train multiple agents offline in parallel, then deploy online the ensemble agent equipped with a higher-resolution pessimism of the Q-ensemble (see Section 4.2 for a more detailed explanation).

4 Method

We propose a simple yet effective framework that addresses the state-action distribution shift described in Section 3. Our method comprises of two parts: (a) a balanced experience replay scheme, and (b) a pessimistic Q-ensemble scheme.

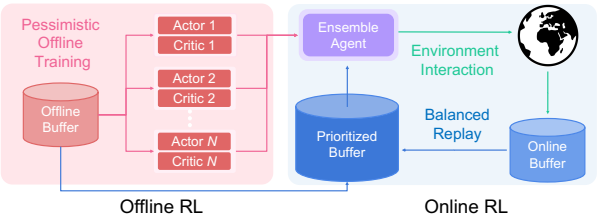


Figure 2: Illustration of our framework. We first train an ensemble of N CQL agents on the offline dataset. Then we fine-tune the ensemble agent using both offline and online transitions via balanced replay. In particular, we train a density ratio estimator that measures the onlineness of a given sample, then store all samples in the prioritized replay buffer with their respective density ratios as priority values. In turn, samples are drawn with probability proportional to their respective priority values.

4.1 Balanced Experience Replay

We introduce a balanced replay scheme that enables us to safely utilize online samples by leveraging relevant, near-on-policy offline samples. By doing so, we can widen the sampling distribution for updates around the on-policy samples and enable timely value propagation. The challenge here is how to design a scheme that locates and retrieves such relevant, near-on-policy samples from the offline dataset, which can often be huge. To achieve this, we measure the *online-ness* of all available samples, and prioritize the samples according to this measure.

In particular, when updating the agent, we propose to sample a transition $(s, a, s') \in \mathcal{B}^{\texttt{off}} \cup \mathcal{B}^{\texttt{on}}$ with a probability proportional to the density ratio $w(s, a) := d^{\texttt{on}}(s, a)/d^{\texttt{off}}(s, a)$ of the given sample. This way, we can retrieve a relevant, near-on-policy sample $(s, a, s') \in \mathcal{B}^{\texttt{off}}$ by locating a transition with high density ratio $w(s, a)$. However, estimating the likelihoods $d^{\texttt{off}}(s, a)$ and $d^{\texttt{on}}(s, a)$ is difficult, since they can in principle be stationary distributions of complex policy mixtures[1]. To avoid this problem, we utilize a likelihood-free density ratio estimation method that estimates $w(s, a)$ by training a network $w_\psi(s, a)$ parametrized by ψ, solely based on samples from $\mathcal{B}^{\texttt{off}}$ and $\mathcal{B}^{\texttt{on}}$.

Training details. Here we describe the training procedure for the density ratio estimator $w_\psi(s, a)$ in detail. Following the idea of Sinha et al. [16], we use the variational representation of f-divergences [17]. Let P and Q be probability measures defined on some measurable space $\mathcal{X}$, with P absolutely continuous w.r.t. Q, and $f(y) := y \log \frac{2y}{y+1} + \log \frac{2}{y+1}$. Then the Jensen-Shannon (JS) divergence is

[1] We remark that $d^{\texttt{off}}(s, a)$ is the stationary distribution of the (arbitrary) behavior policy that generated $\mathcal{B}^{\texttt{off}}$, and $d^{\texttt{on}}(s, a)$ the stationary distribution of the policy that generated $\mathcal{B}^{\texttt{on}}$, which corresponds to the mixture of online policies observed over the course of fine-tuning.

defined as $D_{JS}(P||Q) = \int_{\mathcal{X}} f(dP(x)/dQ(x))dQ(x)$. We then estimate the density ratio dP/dQ with a parametric model $w_\psi(x)$, by maximizing the lower bound of $D_{JS}(P||Q)$ [17]:

$$\mathcal{L}^{\texttt{DR}}(\psi) = \mathbb{E}_{x\sim P}[f'(w_\psi(x))] - \mathbb{E}_{x\sim Q}[f^*(f'(w_\psi(x)))], \tag{4}$$

where $w_\psi(x) \geq 0$ is parametrized by a neural network whose outputs are forced to be non-negative via activation functions, and f^* denotes convex conjugate. In particular, we obtain an estimate $w_\psi(s,a)$ of $d^{\texttt{on}}/d^{\texttt{off}}$ by considering probability distributions P and Q with densities $d^{\texttt{on}}$ and $d^{\texttt{off}}$, respectively. In practice, we sample from $\mathcal{B}^{\texttt{on}}$ for the first term in (4), and from $\mathcal{B}^{\texttt{off}}$ for the latter. For more stable density ratio estimates, we apply self-normalization [18] to the estimated density ratios over $\mathcal{B}^{\texttt{off}}$, similar to Sinha et al. [16]. More details can be found in Section D.

4.2 Pessimistic Q-Ensemble

In order to mitigate distribution shift more effectively, we leverage multiple pessimistically trained Q-functions. We consider an ensemble of N CQL agents pre-trained via update rules (2, 3), i.e., $\{Q_{\theta_i}, \pi_{\phi_i}\}_{i=1}^N$, where θ_i and ϕ_i denote the parameters of the i-th agent's Q-function and policy, respectively. Then we use the ensemble of actor-critic agents whose Q-function and policy are defined as follows:

$$Q_\theta := \frac{1}{N}\sum_{i=1}^N Q_{\theta_i}, \qquad \pi_\phi(\cdot|s) = \mathcal{N}\left(\frac{1}{N}\sum_{i=1}^N \mu_{\phi_i}(s), \quad \frac{1}{N}\sum_{i=1}^N (\sigma^2_{\phi_i}(s) + \mu^2_{\phi_i}(s)) - \mu^2_\phi(s)\right), \tag{5}$$

where $\theta := \{\theta_i\}_{i=1}^N$ and $\phi := \{\phi_i\}_{i=1}^N$. Note that the policy is simply modeled as Gaussian with mean and variance of the Gaussian mixture policy $\frac{1}{N}\sum_{i=1}^N \pi_{\phi_i}$. In turn, θ and ϕ are updated via update rules (1) and (2), respectively, during fine-tuning.

By using a pessimistic Q-function, the agent remains pessimistic with regard to the unseen actions at states encountered online during initial fine-tuning. This is because during early fine-tuning, states resemble those present in the offline dataset, and Q-function generalizes to these states. As we show in our experiments, this protects the good initial policy from severe bootstrap error. And by leveraging multiple pessimistically trained Q-functions, we obtain a more high-resolution pessimism about the unseen data regime. That is, when an individual Q-function may erroneously have high values for unseen samples, Q-ensemble is more robust to these individual errors, and more reliably assigns lower values to unseen samples. Computational overhead of ensemble is discussed in Section B.

5 Related work

Offline RL. Offline RL algorithms aim to train RL agents exclusively with pre-collected datasets. To address the state-conditional action distribution shift, prior methods (a) explicitly constrain the policy to be closed to the behavior policy [2, 3, 4, 5, 19], or (b) train pessimistic value functions [7, 8, 9]. In particular, CQL [9] was used to learn various robotic manipulation tasks [10]. We also build on CQL, so as to leverage pessimism regarding data encountered online during fine-tuning.

Online RL with offline datasets. Several works have explored employing offline datasets for online RL to improve sample efficiency. Some assume access to demonstration data [20, 21, 22, 23, 24], which is limited in that they assume optimality of the dataset. To overcome this, Nair et al. [25] proposed Advantage Weighted Actor Critic (AWAC), which performs regularized policy updates so that the policy stays close to the observed data during both offline and online phases. We instead advocate adopting pessimistic *initialization*, such that we may prevent overoptimism and bootstrap error in the initial online phase, and lift such pessimism once unnecessary, as more online samples are gathered. Some recent works extract behavior primitives from offline data, then learn to compose them online [26, 27, 28]. It would be interesting to apply our method in these setups.

Experience replay. The idea of retrieving important samples for RL was introduced in Schaul et al. [29], where they prioritize samples with high temporal-difference error. The work closest to ours is Sinha et al. [16], which utilizes the density ratios between off-policy and near-on-policy state-action distributions as importance weights for policy evaluation. Our approach differs in that we utilize density ratios for retrieving relevant samples from the offline dataset.

Ensemble methods. In the context of model-free RL, ensemble methods have been studied for addressing Q-function's overestimation bias [30, 31, 32, 33, 34], for better exploration [35, 36, 37], or for reducing bootstrap error propagation [38]. The closest to our approach is Anschel et al. [32] that stabilizes Q-learning by using the average of previously learned Q-values as the target Q-value. While prior works mostly focus on online RL and estimate the ground-truth Q-functions, we leverage an ensemble of pessimistically pre-trained Q-functions for safe offline-to-online RL.

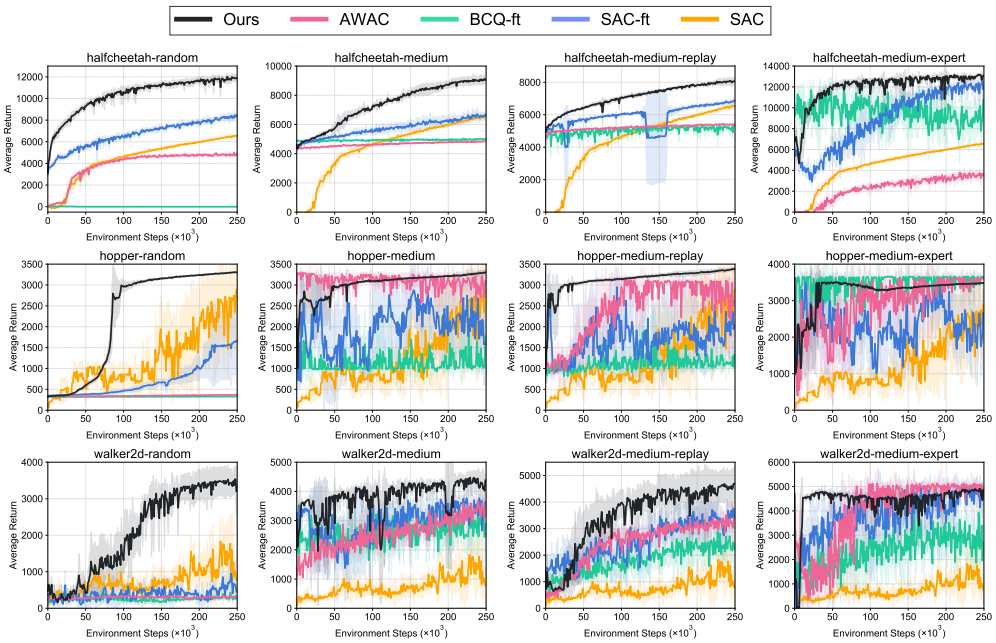


Figure 3: Performance on D4RL [12] MuJoCo locomotion tasks during online fine-tuning. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

6 Experiments

We designed our experiments to answer the following questions:

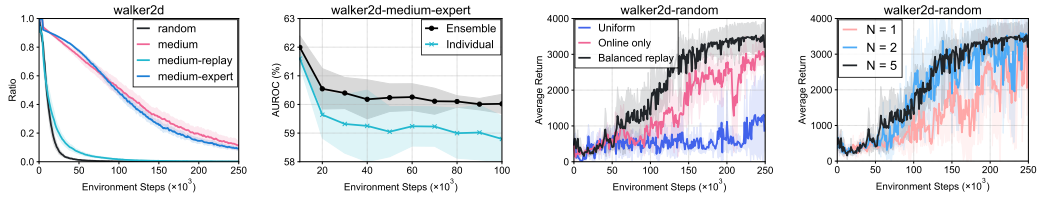
- How does our method compare to existing offline-to-online RL methods and an online RL method that learns from scratch (see Figure 3)?
- Can our balanced replay scheme locate offline samples relevant to the current policy (see Figure 4a) and improve the fine-tuning performance by utilizing these samples (see Figure 4c)?
- Can our pessimistic Q-ensemble scheme discriminate unseen actions (see Figure 4b) and successfully stabilize the fine-tuning procedure by mitigating distribution shift (see Figure 4d)?
- Does our method scale to vision-based robotic manipulation tasks (see Figure 5)?

6.1 Locomotion Tasks

Setup. We consider MuJoCo [11] locomotion tasks, i.e., `halfcheetah`, `hopper`, and `walker2d`, from the D4RL benchmark suite [12]. To demonstrate the applicability of our method on various suboptimal datasets, we use four dataset types: `random`, `medium`, `medium-replay`, and `medium-expert`. Specifically, `random` and `medium` datasets contain samples collected by a random policy and a medium-level policy, respectively. `medium-replay` datasets contain all samples encountered while training a medium-level agent from scratch, and `medium-expert` datasets contain samples collected by both medium-level and expert-level policies. For our method, we use ensemble size $N = 5$. More experimental details are provided in Section D.

Comparative Evaluation. We consider the methods outlined below as baselines for comparative evaluation. For fair comparison, we applied ensemble to all baselines except SAC-ft, since the results for SAC-ft with ensemble can be found in the ablation studies (see Figure 4c).

- Advantage Weighted Actor Critic [AWAC; 25]: an offline-to-online RL method that trains the policy to imitate actions with high advantage estimates.
- BCQ-ft: Batch-Constrained deep Q-learning [BCQ; 2], is an offline RL method that updates policy by modeling the behavior policy using a conditional VAE [39]. We extend BCQ to the online fine-tuning setup by applying the same update rules as offline training.
- SAC-ft: Starting from a CQL agent trained via (3, 2), we fine-tune the agent via SAC updates (1, 2). Justification for excluding the CQL regularization term from (3) during fine-tuning can be found in Section E.1.
- SAC: a SAC agent trained from scratch via (1, 2), i.e., the agent has no access to the offline dataset. This baseline highlights the benefit of offline-to-online RL, as opposed to fully online RL, in terms of sample efficiency.



(a) Buffer analysis (b) Ensemble analysis (c) Buffer ablation (d) Ensemble ablation

Figure 4: (a) Proportion of offline samples used for updates as the agent is fine-tuned online, for `walker2d` tasks. (b) AUROC (%) over the course of fine-tuning on `walker2d-medium-expert`, where the Q-function is interpreted as a binary classifier that classifies a given state-action pair (s, a) as either a seen pair $(s, a_{\texttt{seen}})$ or an unseen pair $(s, a_{\texttt{uniform}})$, for a state s encountered online. Pessimistic Q-ensemble shows a stronger discriminative ability. (c) Performance on `walker2d-random` with and without balanced experience replay. We consider two setups where balanced experience replay is not used: (i) Uniform, where offline and online samples are sampled uniformly from the same buffer for updates, and (ii) Online only, where the offline agent is fine-tuned using online samples only. (d) Performance on `walker2d-random` with varying ensemble size $N \in \{1, 2, 5\}$. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

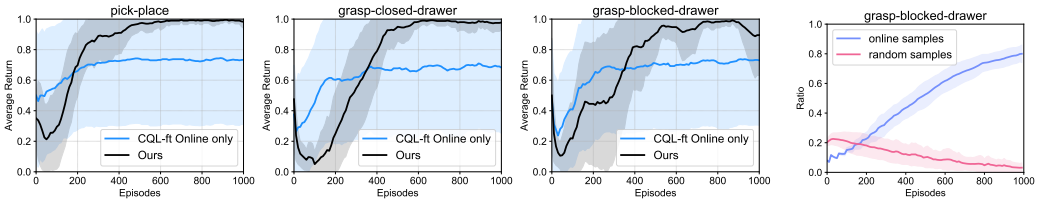
Figure 3 shows the performances of our method and baseline methods considered during the online RL phase. In most tasks, our method outperforms the baseline methods in terms of both sample-efficiency and final performance. In particular, our method significantly outperforms SAC-ft, which shows that balanced replay and pessimistic Q-ensemble are indeed essential.

We also emphasize that our method performs consistently well across all tasks, while the performances of AWAC and BCQ-ft are highly dependent on the quality of the offline dataset. For example, we observe that AWAC and BCQ-ft show competitive performances in tasks where the datasets are generated by high-quality policies, i.e., `medium-expert` tasks, but perform worse than SAC on `random` tasks. This is because AWAC and BCQ-ft employ the same regularized, pessimistic update rule for offline and online setups alike, either explicitly (BCQ-ft) or implicitly (AWAC), which leads to slow fine-tuning. Our method instead relies on pessimistic *initialization*, and hence enjoys much faster fine-tuning, while not sacrificing the initial training stability.

Balanced replay analysis. To investigate the effectiveness of our balanced experience replay scheme for locating near-on-policy samples in the offline dataset, we report the ratios of offline samples used for updates fine-tuning proceeds. Figure 4a shows that for the `random` task, offline samples quickly become obsolete, as they quickly become irrelevant to the policy being fine-tuned. However, for the `medium-expert` task, offline samples include useful expert-level transitions that are relevant to the current policy, hence are replayed throughout the online training. This shows that our balanced replay scheme is capable of utilizing offline samples only when appropriate.

Q-ensemble analysis. We quantitatively demonstrate that pessimistic Q-ensemble indeed provides more discriminative value estimates, i.e., having distinguishably lower Q-values for unseen actions than for seen actions. In particular, we consider a `medium-expert` dataset, where the offline data distribution is narrow, and the near-optimal offline policy can be brittle. Let $\mathcal{D}_T^{\texttt{real}} := \{(s_i, a_i)\}_{i=1}^T$ be the samples collected online up until timestep T. We construct a "fake" dataset by replacing the actions in $\mathcal{D}_T^{\texttt{real}}$ with random actions, i.e., $\mathcal{D}_T^{\texttt{fake}} := \{(s_i, a_{\texttt{unif}})\}_{i=1}^T$. Interpreting $Q(s, a)$ as the confidence value for classifying real and fake transitions, we measure the area under ROC (AUROC) curve values over the course of fine-tuning. As seen in Figure 4b, Q-ensemble demonstrates superior discriminative ability, which leads to stable fine-tuning.

Ablation studies. Figure 4c shows that balanced replay improves fine-tuning performance by sampling near-on-policy transitions. On the other hand, two other naïve sampling schemes – (a) Uniform, where offline and online samples are sampled uniformly from the same buffer, and (b) Online only, where the offline agent is fined-tuned using online samples exclusively – suffer from slow and unstable improvement, even with pessimistic Q-ensemble. This shows that balanced replay is crucial for reducing the harmful effects of distribution shift.



(a) Performance (b) Buffer analysis

Figure 5: (a) Fine-tuning performance for robotic manipulation tasks considered. (b) Proportion of random data used for during fine-tuning decreases over time. The solid lines and shaded regions represent mean and standard deviation, respectively, across eight runs.

Also, Figure 4d shows that fine-tuning performance improves as the ensemble size N increases, which shows that larger ensemble size provides higher-resolution pessimism, leading to more stable policy updates. Ablation studies for all tasks can be found in Section E.1.

6.2 Robotic Manipulation Tasks

Setup. We consider three sparse-reward pixel-based manipulation tasks from Singh et al. [10]: (1) `pick-place`: pick an object and put it in the tray; (2) `grasp-closed-drawer`: grasp an object in the initially closed bottom drawer; (3) `grasp-blocked-drawer`: grasp an object in the initially closed bottom drawer, where the initially open top drawer blocks the handle for the bottom drawer. Episode lengths for the tasks are 40, 50, 80, respectively.

The original dataset [10] for each task consists of scripted exploratory policy rollouts. For example, for `pick-place`, the dataset contains scripted pick attempts and place attempts. However, it is rarely the case that logged data 'in the wild' contains such structured, high-quality transitions only. We consider a more realistic setup where the dataset also includes undirected, exploratory samples – we replace a subset of the original dataset with uniform random policy rollouts. Note that random policy rollouts are common in robotic tasks [40, 41]. We used ensemble size $N = 4$ for our method. More details about the tasks and dataset construction are provided in Section D.2.

Comparative Evaluation. We compare our method with the method considered in Singh et al. [10], namely, CQL fine-tuning with online samples only. CQL-ft fails to solve the task in some of the seeds, resulting in high variance as seen in Figure 5a. This is because CQL shows inconsistent offline performance across random seeds due to such factors as difficulty of training on mixture data [12], instability of offline agents over stopping point of training [42], and sparsity of rewards. With no access to (pseudo-)expert offline data and due to heavy regularization of CQL, such CQL agents hardly improve. Meanwhile, our method consistently learns to perform the task within a reasonable amount of additional environment interaction (40K to 80K steps).

Buffer analysis. We analyze whether balanced replay scales to robotic tasks based on image observations. As seen in Figure 5b, without any privileged information, balanced replay automatically selects relevant offline samples for updates, while filtering out task-irrelevant, random data as fine-tuning proceeds.

7 Conclusion

In this paper, we identify state-action distribution shift as the major obstacle in offline-to-online RL. To address this, we present a simple framework that incorporates (1) a balanced experience replay scheme, and (2) a pessimistic Q-ensemble scheme. Our experiments show that the proposed method performs well across many continuous control robotic tasks, including locomotion and manipulation tasks. We expect our method to enable more sample-efficient training of robotic agents by leveraging offline samples both for offline and online learning. We also believe our method could prove to be useful for other relevant topics such as scalable RL [43] and RL safety [44].

References

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

[2] S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, 2019.

[3] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, 2019.

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

[5] N. Y. Siegel, J. T. Springenberg, F. Berkenkamp, A. Abdolmaleki, M. Neunert, T. Lampe, R. Hafner, and M. Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. In *International Conference on Learning Representations*, 2020.

[6] R. Agarwal, D. Schuurmans, and M. Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, 2020.

[7] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims. Morel: Model-based offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

[8] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization. In *Advances in Neural Information Processing Systems*, 2020.

[9] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

[10] A. Singh, A. Yu, J. Yang, J. Zhang, A. Kumar, and S. Levine. Cog: Connecting new skills to past experience with offline reinforcement learning. In *Conference on Robot Learning*, 2020.

[11] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, 2012.

[12] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[13] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018.

[14] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

[15] T. L. Paine, C. Paduraru, A. Michi, C. Gulcehre, K. Zolna, A. Novikov, Z. Wang, and N. de Freitas. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*, 2020.

[16] S. Sinha, J. Song, A. Garg, and S. Ermon. Experience replay with likelihood-free importance weights, 2021. URL https://openreview.net/forum?id=ioXEbG_Sf-a.

[17] X. Nguyen, M. J. Wainwright, and M. I. Jordan. Estimating divergence functionals and the likelihood ratio by penalized convex risk minimization. In *Advances in Neural Information Processing Systems*, 2008.

[18] W. G. Cochran. *Sampling techniques*. John Wiley & Sons, 2007.

[19] S. K. S. Ghasemipour, D. Schuurmans, and S. S. Gu. Emaq: Expected-max q-learning operator for simple yet effective offline and online rl. *arXiv preprint arXiv:2007.11091*, 2020.

[20] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems*, 2003.

[21] B. Kim, A.-m. Farahmand, J. Pineau, and D. Precup. Learning from limited demonstrations. In *Advances in Neural Information Processing Systems*, 2013.

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

[23] H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. In *International Conference on Robotics and Automation*, 2019.

[24] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[25] A. Nair, M. Dalal, A. Gupta, and S. Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2021.

[26] K. Pertsch, Y. Lee, and J. J. Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on Robot Learning*, 2020.

[27] A. Ajay, A. Kumar, P. Agrawal, S. Levine, and O. Nachum. {OPAL}: Offline primitive discovery for accelerating offline reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=V69LGwJ0lIN.

[28] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine. Parrot: Data-driven behavioral priors for reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Ysuv-WOFeKR.

[29] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016.

[30] H. V. Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, 2010.

[31] H. v. Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI Conference on Artificial Intelligence*, 2016.

[32] O. Anschel, N. Baram, and N. Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *International Conference on Machine Learning*, 2017.

[33] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, 2018.

[34] Q. Lan, Y. Pan, A. Fyshe, and M. White. Maxmin q-learning: Controlling the estimation bias of q-learning. In *International Conference on Learning Representations*, 2020.

[35] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, 2016.

[36] R. Y. Chen, S. Sidor, P. Abbeel, and J. Schulman. Ucb exploration via q-ensembles. *arXiv preprint arXiv:1706.01502*, 2017.

[37] K. Lee, M. Laskin, A. Srinivas, and P. Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. In *International Conference on Machine Learning*, pages 6131–6141. PMLR, 2021.

[38] K. Lee, M. Laskin, A. Srinivas, and P. Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. *arXiv preprint arXiv:2007.04938*, 2020.

[39] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, 2015.

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

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

[42] S. Fujimoto and S. Gu. A minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2106.06860*, 2021.

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

[44] J. Garcıa and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[45] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[46] Y. Burda, R. Grosse, and R. Salakhutdinov. Importance weighted autoencoders. In *International Conference on Learning Representations*, 2016.

Supplementary Material

A Pseudocode

We provide a pseudocode for the entire training procedure. For conciseness of presentation, we henceforth let $\tau = (s, a, r, s')$ denote a transition stored in the replay buffer.

Algorithm 1 Offline RL

Inputs: Ensemble size N, offline dataset $\mathcal{B}^{\texttt{off}}$

Initialize parameters of N independent CQL agents $\{Q_{\theta_i}, \pi_{\phi_i}\}_{i=1}^{N}$
for $i = 1, \cdots, N$ **do**
  for each training iteration **do**
    Sample a random minibatch $\{\tau_j\}_{j=1}^{B} \sim \mathcal{B}^{\texttt{off}}$
    Calculate $\mathcal{L}_{\texttt{critic}}^{\texttt{CQL}}(\theta_i)$ and update θ_i
    Calculate $\mathcal{L}_{\texttt{actor}}^{\texttt{SAC}}(\phi_i)$ and update ϕ_i
  end for
end for
// ENSEMBLE AGENT
Define Q-ensemble $Q_\theta := \frac{1}{N}\sum_{i=1}^{N} Q_{\theta_i}$
Define $\pi_\phi(\cdot|s) := \mathcal{N}\left(\frac{1}{N}\sum_{i=1}^{N}\mu_{\phi_i}(s), \quad \frac{1}{N}\sum_{i=1}^{N}(\sigma_{\phi_i}^2(s) + \mu_{\phi_i}^2(s)) - \mu_\phi^2(s)\right)$
Return Ensemble agent $\{Q_\theta, \pi_\phi\}$

Algorithm 2 Online RL

Inputs: Ensemble agent $\{\pi_\phi, Q_\theta\}$, offline buffer $\mathcal{B}^{\texttt{off}}$

Initialize parameters of density ratio estimation network, ψ
Initialize online buffer $\mathcal{B}^{\texttt{on}} \leftarrow \emptyset$, prioritized buffer $\mathcal{B}^{\texttt{priority}} \leftarrow \emptyset$, and default priority value $p_0 \leftarrow 1.0$
for $j = 1, \ldots, |\mathcal{B}^{\texttt{off}}|$ **do**
  $\mathcal{B}^{\texttt{priority}} \leftarrow \mathcal{B}^{\texttt{priority}} \cup \{(\tau_j, p_0)\}$
end for
$p_0 \leftarrow P_0$ (see Training details for balanced replay in Section D for explanation)
for each iteration **do**
  // COLLECT TRAINING SAMPLES
  Collect a transition $\tau = (s, a, r, s')$ via environment interaction with π_ϕ
  Update $\mathcal{B}^{\texttt{on}} \leftarrow \mathcal{B}^{\texttt{on}} \cup \{\tau\}$, $\mathcal{B}^{\texttt{priority}} \leftarrow \mathcal{B}^{\texttt{priority}} \cup \{(\tau, p_0)\}$
  // UPDATE DENSITY RATIO ESTIMATION NETWORK
  Sample $\{\tau_i^{\texttt{off}}\}_{i=1}^{B} \sim \mathcal{B}^{\texttt{off}}$, $\{\tau_i^{\texttt{on}}\}_{i=1}^{B} \sim \mathcal{B}^{\texttt{on}}$
  Calculate $\mathcal{L}^{\texttt{DR}}(\psi)$ and update ψ
  // UPDATE AGENT AND PRIORITY VALUES
  Sample a random minibatch $\{\tau_j\}_{j=1}^{B} \sim \mathcal{B}^{\texttt{priority}}$
  Calculate $\mathcal{L}_{\texttt{critic}}^{\texttt{SAC}}(\theta)$ and update θ
  Calculate $\mathcal{L}_{\texttt{actor}}^{\texttt{SAC}}(\phi)$ and update ϕ
  for $j = 1, \ldots, B$ **do**
    Update priority of τ_j to be $\frac{w(s_j, a_j)}{\mathbb{E}_{\tau^{\texttt{off}} \sim \mathcal{B}^{\texttt{off}}}[w(s,a)]}$ (see Training details for balanced replay in Section D for explanation)
    $p_0 \leftarrow \max(p_0, \frac{w(s_j, a_j)}{\mathbb{E}_{\tau^{\texttt{off}} \sim \mathcal{B}^{\texttt{off}}}[w(s,a)]})$
  end for
end for

B Computational Complexity

Ensemble incurs additional computation overhead for training and inference. A naïve implementation with ensemble size N leads to $N\times$ training and inference time. However, it is possible to parallelize the networks (which we did for MuJoCo locomotion tasks) for computational efficiency.

C Training Details for Concept Experiments

VAE training. For generating the plot in Figure 1a, we first split the `halfcheetah-random` offline dataset consisting of random halfcheetah transitions into training and validation datasets. Then we trained a variational autoencoder on the training dataset using the Adam optimizer [45] with learning rate $1\mathrm{e}{-3}$. We estimated the log-likelihood of (a) online samples gathered by an offline agent trained via CQL (3, 2), and (b) offline samples in the validation dataset, using the importance weight estimator [46].

Sample selection analysis. For generating the plot in Figure 1b, we first trained a CQL agent Q_θ, π_ϕ on the `halfcheetah-medium` dataset consisting of medium-quality halfcheetah transitions. Then, we fine-tune the CQL agent with two different strategies: Uniform and Online only. First, Uniform refers to the strategy where we keep a single replay buffer, in which we store both offline and online samples, then sampling a minibatch from the replay buffer uniformly at random, i.e., $\{\tau_j\}_{j=1}^B \sim \mathcal{B}^{\texttt{off}} \cup \mathcal{B}^{\texttt{on}}$. Here, the sampling procedure is agnostic of whether the sample is offline or online. On the other hand, Online only refers to the strategy that exploits online samples only, i.e., $\{\tau_j\}_{j=1}^B \sim \mathcal{B}^{\texttt{on}}$.

Choice of offline Q-function. For generating the plots in Figure 1c, we considered the `halfcheetah-random` and `halfcheetah-medium-expert` offline datasets, each consisting of random, and a mixture of medium and expert transitions, respectively. For a given offline dataset $\mathcal{B}^{\texttt{off}}$, we first trained a CQL agent $\{Q_{\theta_{\texttt{CQL}}}, \pi_\phi\}$ via (3, 2). Then, with π_ϕ and $\mathcal{B}^{\texttt{off}}$, we trained a non-pessimistic $Q_{\theta_{\texttt{FQE}}}$ via Fitted Q-Evaluation [FQE; 15]. To elaborate, given a fixed policy π_ϕ, FQE trains a Q-function starting from random initialization by performing policy evaluation (Bellman backup) until convergence. In practice, we used the Adam optimizer [45] with learning rate $3\mathrm{e}{-4}$ and batch size $B = 256$, and trained for 250k iterations. In the online fine-tuning phase, we fine-tuned $\{\pi_\phi, Q_{\theta_{\texttt{CQL}}}\}$ and $\{\pi_\phi, Q_{\theta_{\texttt{FQE}}}\}$, respectively, via SAC update rules (1, 2). We applied the balanced replay scheme introduced in Section 4.1, in order to facilitate fine-tuning.

D Experimental Setup Details

D.1 Locomotion Tasks

Figure 6: Locomotion tasks. `halfcheetah`, `hopper`, and `walker2d`, in order.

Task detail & offline dataset. For training offline RL agents, we used the publicly available datasets from the D4RL benchmark suite (https://github.com/rail-berkeley/d4rl) without any modification to the datasets. Specifically, we consider three MuJoCo locomotion tasks, namely, `halfcheetah`, `hopper`, and `walker2d`. The goal of each task is to move forward as fast as possible, while keeping the control cost minimal. For each task, we consider four types of datasets:

- `random`: contains random policy rollouts;
- `medium`: contains medium-level policy rollouts;
- `medium-replay`: contains all samples seen while training a medium-level agent from scratch;
- `medium-expert`: half of the dataset consists of medium-level policy rollouts and the other half consists of expert-level policy rollouts.

Training details for offline RL. For training CQL agents, following Kumar et al. [9], we built CQL on top of the publicly available implementation of SAC (https://github.com/vitchyr/rlkit). As for network architecture, we use 2-layer multi-layer perceptrons (MLPs) for value and policy networks (except for `halfcheetah-medium-expert` task where we found that 3-layer MLPs is more effective for training CQL agents). For all locomotion experiments, following the setup in Kumar et al. [9], we trained offline RL agents for 1000 epochs without early stopping.

Training details for online RL. For all locomotion experiments, we report the fine-tuning performance during 250K timesteps for 4 random seeds. For our method, we used Adam optimizer [45] with policy learning rates chosen from $\{3\text{e}{-4}, 3\text{e}{-5}, 5\text{e}{-6}\}$, and value learning rate of $3\text{e}{-4}$. We used ensemble size $N = 5$ We found that taking 5 $\times$ training steps (5000 as opposed to the usual 1000) after collecting the first 1000 online samples slightly improves the fine-tuning performance. After that, we trained for 1000 training steps every time 1000 additional samples were collected.

Training details for balanced replay. For training the density ratio estimation network $w_\psi(s, a)$, which was fixed to be a 2-layer MLP for all our experiments, we used batch size 256 (i.e., 256 offline samples and 256 online samples), and learning rate $3\text{e}{-4}$ for all locomotion experiments. When calculating the priority values, we applied self-normalization:

$$\widetilde{w}_\psi(x) = \frac{w_\psi(x)^{1/T}}{\mathbb{E}_{x\sim P}[w_\psi(x)^{1/T}]}, \tag{6}$$

where x and P denote (s, a) and $\mathcal{B}^{\texttt{off}}$, respectively, and T is the temperature hyperparameter. For all locomotion experiments, we used $T = 5.0$. This stabilizes training by producing stable density ratios across random minibatches.

Before fine-tuning starts, we first add the offline samples to the prioritized replay buffer with default priority value 1.0. Then, we set the default priority value to be high, in order to make sure online samples are seen enough for RL updates. In particular, letting M denote the size of the offline buffer, we set the default priority value to be such that the initial 1000 online samples gathered would have probability ρ of being seen, where ρ is a hyperparameter, i.e., priority value of $P_0 := \frac{M}{1000} \cdot \frac{\rho}{1-\rho}$. We used $\rho \in \{0.5, 0.75\}$.

Note that after being seen once for an RL update, the given sample's priority value is appropriately updated to be $\tilde{w}_\psi(s, a)$. Afterwards, the default priority value is updated to be the maximum priority value seen during fine-tuning, as in [29]. See Section A for a pseudocode.

Baselines. For AWAC [25], we use the publicly available implementation from the authors (https://github.com/vitchyr/rlkit/tree/master/examples/awac) without any modification to hyperparameters and architectures. For training BCQ [2], we use the publicly available implementation from https://github.com/rail-berkeley/d4rl_evaluations/tree/master/bcq. For the online setup (BCQ-ft), we used additional online samples for training. For SAC [13], we used the implementation from rlkit (https://github.com/vitchyr/rlkit) with default hyperparameters.

D.2 Robotic Manipulation Tasks

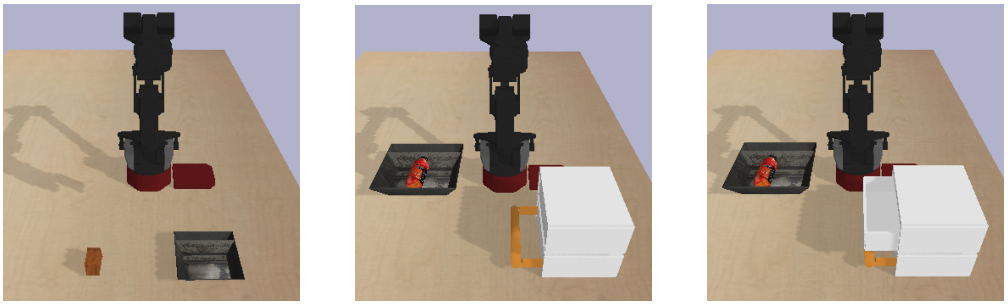


Figure 7: Manipulation tasks. (Left) `pick-place`, (Middle) `grasp-closed-drawer`, (Right) `grasp-blocked-drawer`.

Task detail & offline dataset. We consider three sparse-reward pixel-based manipulation tasks from Singh et al. [10] (see Figure 7):

- `pick-place`: pick an object and put it in the tray.
- `grasp-closed-drawer`: grasp an object in the initially closed bottom drawer;
- `grasp-blocked-drawer`: grasp an object in the initially closed bottom drawer, where the initially open top drawer blocks the handle for the bottom drawer.

Episode lengths for the tasks are 40, 50, 80, respectively.

For each task, the offline dataset provided by the authors of Singh et al. [10] (https://github.com/avisingh599/cog) consists of highly structured transitions. For example, for `pick-place`, the dataset contains grasp attempts from a randomized scripted policy and place attempts from a randomized scripted policy; for `grasp-closed-drawer` and `grasp-blocked-drawer`, dataset contains various behaviors such as opening and closing both drawers, grasping object in the scene, and placing objects at random places. For a more detailed description, refer to Singh et al. [10]. Meanwhile, it is rarely the case that logged data 'in the wild' are generated by such structured, directed policies only. To consider a more realistic setup, we replace half of the original dataset with 100K uniform random policy rollouts. We remark that uniform random policy rollouts are commonly used in robotic tasks [40, 41].

Training details for offline RL. We used the official implementation by the authors of Singh et al. [10] (https://github.com/avisingh599/cog) for training offline CQL agents on the modified datasets, with default hyperparameters. Specifically, the value network is parametrized by three convolutional layers and two pooling layers in-between, followed by fully-connected layers of hidden sizes (1024, 512, 256) with ReLU activations. Kernels of size 3×3 with 16 channels are used for all three convolutional layers. Maxpooling is applied after each of the first two convolutional layers, with stride 2. The policy network is identical to the value network, except it does not take action as an input. Policy learning rate and value learning rate are $1\mathrm{e}{-4}$ and $3\mathrm{e}{-4}$, respectively. For each task, we trained the agent for 500K train steps, then picked the best performing checkpoint in terms of average performance across seeds.

Training details for online RL. For our main results, we report fine-tuning performance during 1K episodes for 8 random seeds. For our method, we used Adam optimizer [45] with policy learning rate $3\mathrm{e}{-5}$ and value learning rate $1\mathrm{e}{-4}$. Again, we applied $5 \times$ training steps after collecting the first batch of samples. Architecture for density ratio estimation network w_ψ is identical to the value network, except it has fully-connected layers with hidden sizes (256, 128). We used learning rate $3\mathrm{e}{-4}$ for w_ψ and temperature $T = 2.5$ for balanced replay in all manipulation tasks. We used ensemble size $N = 4$.

Baseline. For CQL-online, we used the official implementation by the authors of Singh et al. [10] (https://github.com/avisingh599/cog). Specifically, the same CQL regularization is applied during online fine-tuning, while using online samples exclusively for updates. Learning rates are the same as in offline training.

E Additional Experiment

E.1 Locomotion

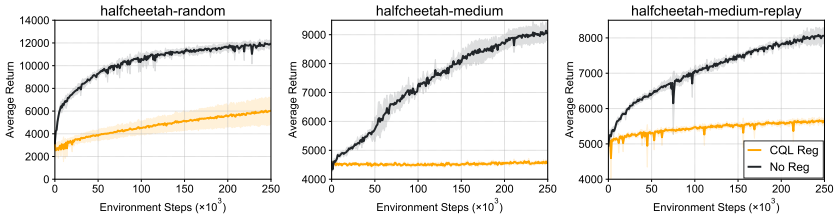


Figure 8: Fine-tuning performance with CQL regularization for `halfcheetah` tasks. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

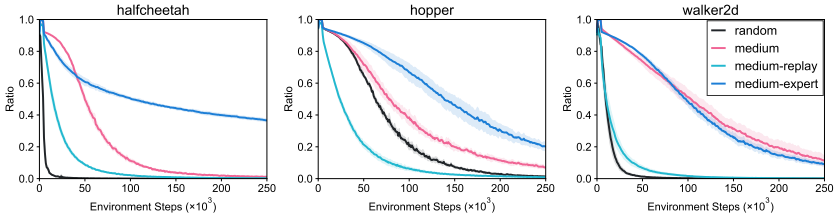


Figure 9: Proportion of offline samples used for updates during fine-tuning. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

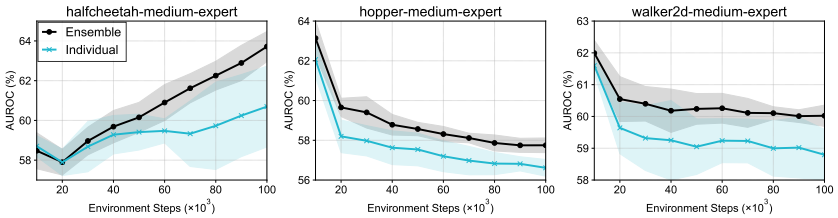


Figure 10: AUROC (%) over the course of fine-tuning on `medium-expert` tasks, where the Q-function is interpreted as a binary classifier that classifies a given state-action pair (s, a) as either a seen pair $(s, a_{\texttt{seen}})$ or an unseen pair $(s, a_{\texttt{uniform}})$, for a state s encountered online. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

Effects of CQL Regularization on Fine-tuning. Here, we show that removing the regularization term from the CQL objective $\mathcal{L}^{\texttt{CQL}}_{\texttt{critic}}$ is crucial for fast fine-tuning. After training an offline RL agent via CQL (3, 2), we fine-tuned the agent via CQL objective (3) (denoted CQL Reg), and via SAC objective (1) (denoted No reg), respectively. For both methods, balanced replay was applied. As shown in Figure 8, fine-tuning with CQL regularization prevents the agent from improving via online interaction, for pessimistic updates during fine-tuning (as opposed to pessimistic *initialization*) essentially results in failure to explore.

Balanced replay analysis. We provide buffer analysis as in Figure 4a for all locomotion tasks. As seen in Figure 9, trends are similar across all environments, i.e., balanced replay automatically decides whether offline samples will be used or not depending their relevance to the current agent.

Q-ensemble analysis. We provide further details and experiments for Q-ensemble analysis from Figure 4b. Letting $\mathcal{D}_T^{\texttt{real}} := \{(s_i, a_i)\}_{i=1}^T$ be the samples collected online up until timestep T, we construct a corresponding fake dataset by replacing the actions in $\mathcal{D}_T^{\texttt{real}}$ by uniform random actions, i.e., $\mathcal{D}_T^{\texttt{fake}} := \{(s_i, a_{\texttt{unif}})\}_{i=1}^T$. Note that to provide enough coverage of the action space, we sampled 50 random actions for each real state-action pair considered. Then, interpreting $Q(s, a)$ as the confidence value for classifying real and fake transitions, we measure the area under ROC (AUROC) curve values over the course of fine-tuning. As seen in Figure 10, Q-ensemble demonstrates superior discriminative ability, which prevents distribution shift and leads to stable fine-tuning.

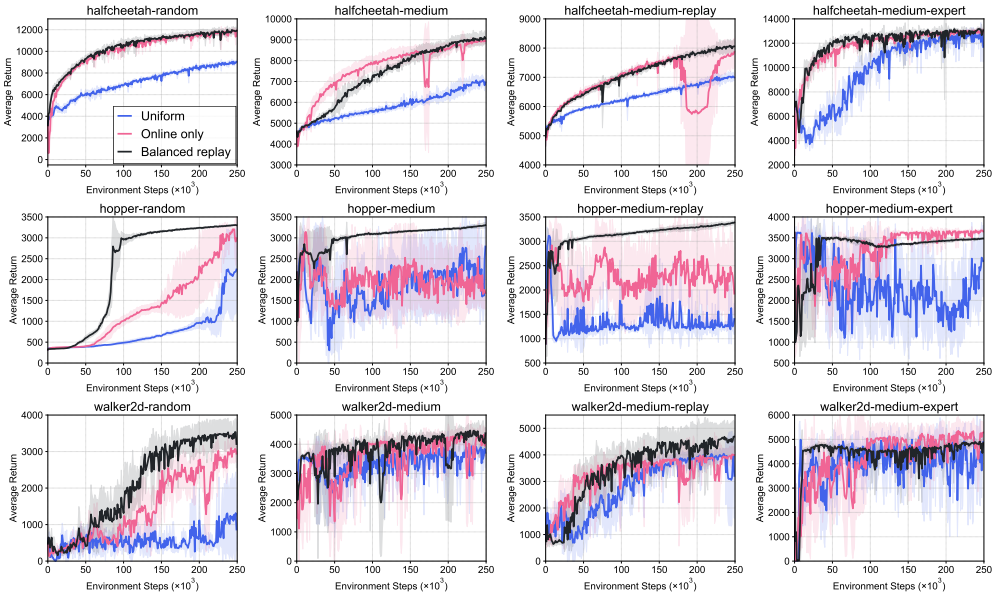


Figure 11: Ablation on balanced replay. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

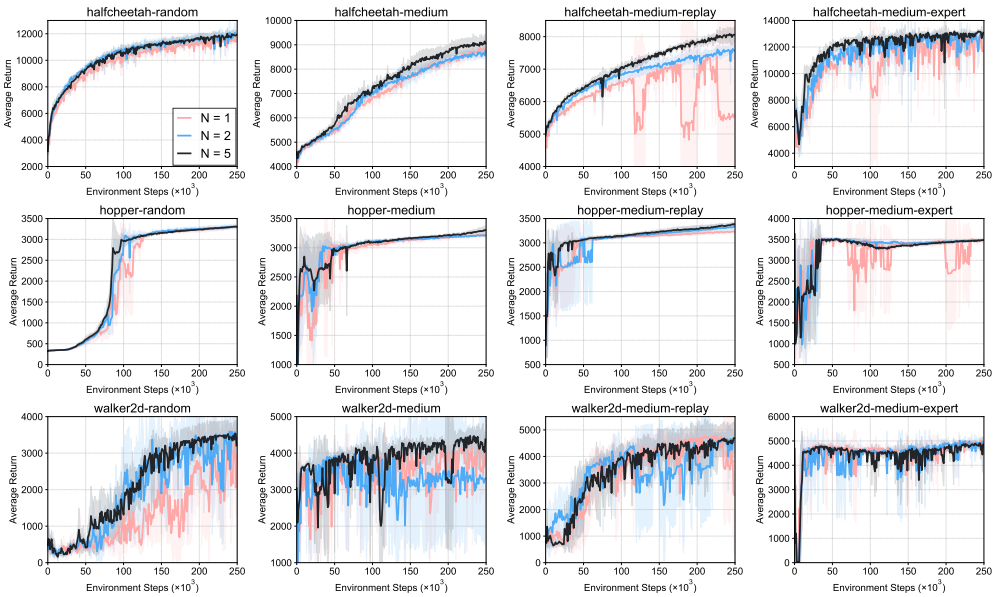


Figure 12: Fine-tuning performance with varying ensemble sizes. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

Ablation on balanced replay. We provide ablation study on balanced replay as in Figure 4c for all tasks (Figure 11). Balanced replay provides significant performance gain compared to the other two sampling strategies: Online only, and Uniform. The benefit is especially pronounced in the more complex environment of `hopper` and `walker2d`.

Ablation on ensemble size. We provide learning curves for all tasks with varying ensemble size $N \in \{1, 2, 5\}$. As seen in Figure12, performance improves as N grows. However, due to the trade-off between computational overhead and performance, we opted for $N = 5$ for our method.

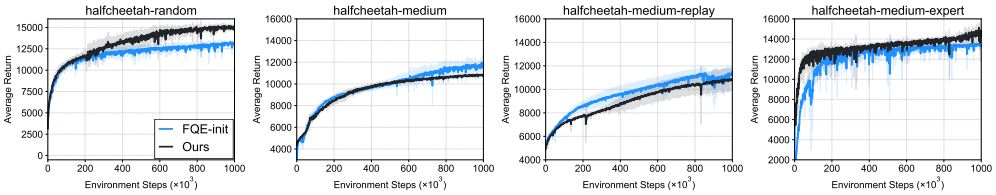


Figure 13: Asymptotic performance of our method (pessimistic Q-function) and FQE-init that fine-tunes a neutral Q-function. Ensemble and balanced replay were applied to both methods. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

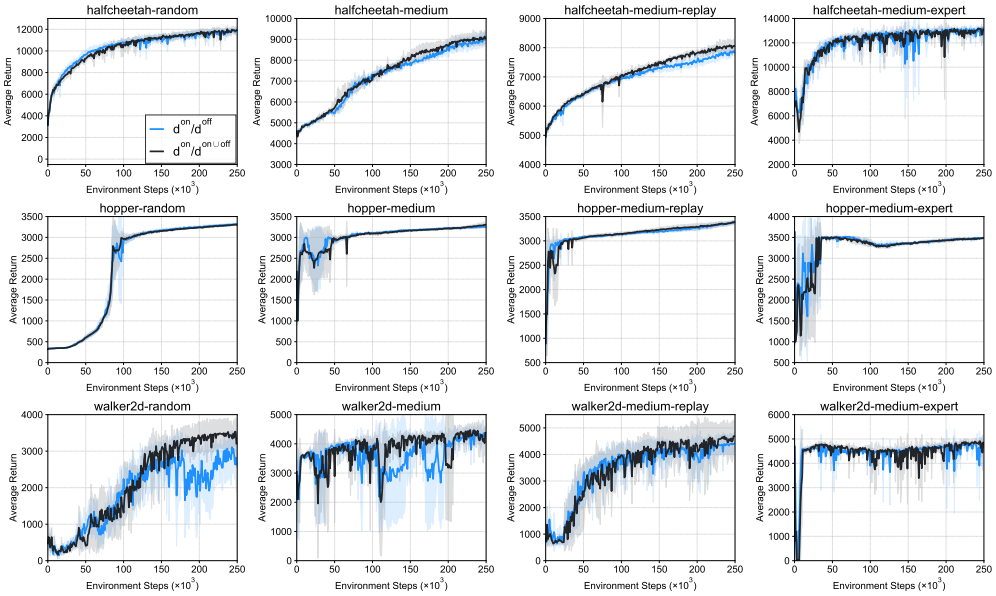


Figure 14: An alternative density ratio ($d^{\texttt{on}}/d^{\texttt{on}\cup\texttt{off}}$) for balanced replay. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

Dataset composition and performance. In our experiments, we observed that dataset composition may be related to the fine-tuning performance in a counterintuitive way. In order to investigate this, we additionally ran our method on halfcheetah tasks for 1 million steps. As shown in Figure 13, our method performed best on random dataset, reaching 15k average return, while reaching only 12k for medium and medium-replay datasets. We conjecture this is because the agent sees the most diverse set of samples when trained from scratch. This means that Q-function can extrapolate better, and the agent can explore more efficiently as a result. While the medium-replay dataset also contains diverse samples, it asymptotically performs worst, possibly because it contains significantly less data compared to the other datasets – random and medium contain 1 million transitions, respectively, and medium-expert contains 2 million transitions.

We note that failure to explore properly is not necessarily due to the pessimism in Q-function, but is rather a legacy of the dataset composition. To show this, we fine-tuned an ensemble of FQE-initialized agents (neutral Q-function) with balanced replay (see Section 3.3 and Section C for details) then compared it with our method (pessimistic Q-function), as in Section 3.3. As shown in Figure 13, we observe that the two methods show more or less similar asymptotic performances with the exception of halfcheetah-random, where our method noticeably outperforms FQE-init. This shows that initial pessimism does not harm the asymptotic online performance, while preventing initial performance degradation in many cases (meanwhile, FQE-init suffers from performance degradation for medium and medium-expert tasks, a result in line with the observation from Section 3.3).

Exploring different density ratio schemes. We also additionally experimented with density ratio $w(s,a) := d^{\texttt{on}}(s,a)/d^{\texttt{off}\cup\texttt{on}}(s,a)$ instead of $w(s,a) := d^{\texttt{on}}(s,a)/d^{\texttt{off}}(s,a)$ for balanced replay. The hyperparameters were set identically to the original setup. As seen in Figure 14, we observed similar performance across most setups. However, the alternative scheme may have the potential benefit of trending towards 1.0 asymptotically.

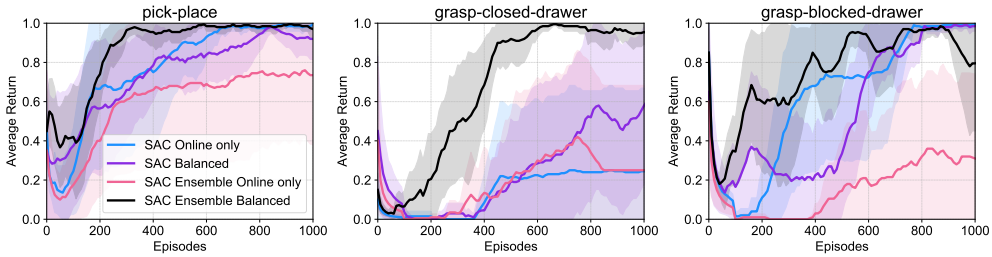


Figure 15: Ablation study with various ablated versions of our method, but with SAC objective. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

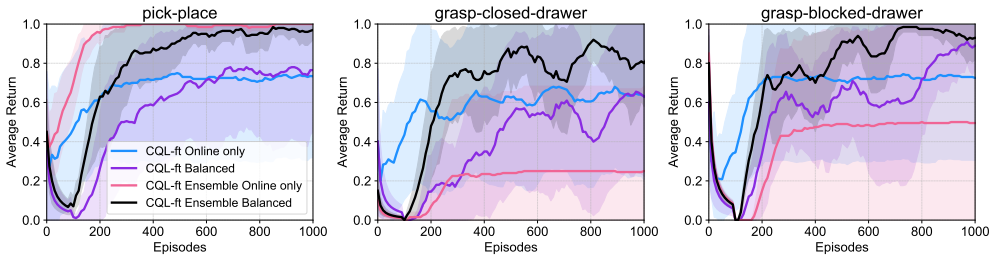


Figure 16: Ablation study with various ablated versions of our method, but with CQL objective. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

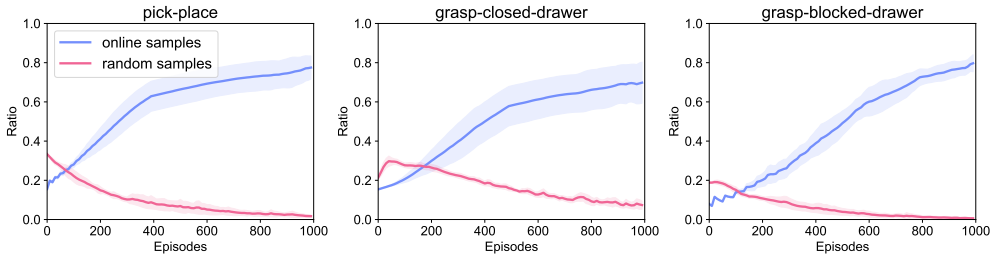


Figure 17: Proportion of random data used for during fine-tuning decreases over time. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

E.2 Manipulation

Ablation studies. We provide ablation studies of our method for the robotic manipulation tasks (Figure 15). Starting from offline CQL agents, we consider:

- SAC Online only: Fine-tune with SAC, with online samples only;
- SAC Balanced: Fine-tune with SAC, with balanced replay;
- SAC Ensemble Online only: Fine-tune an ensemble agent with SAC, with online samples only;
- SAC Ensemble Balanced: Ours, i.e., fine-tune an ensemble agent with SAC and balanced replay.

Furthermore, we additionally experimented with CQL objectives instead of SAC objectives, and provide the results (Figure 16).

In most cases, both ensemble and balanced replay were essential for good performance. First, due to reward sparsity, there were individual offline agents that, while producing meaningful trajectories that resemble some trajectories in the offline dataset, were nonetheless not good enough to receive reward signals online. In this case, the ensemble agent becomes more robust than these individual agents, and was more likely to see enough reward signals for fine-tuning. Second, even with ensemble, updating with online samples only resulted in severe distribution shift, and balanced replay was essential to facilitate fine-tuning. However, note that it only took our method about 200 episodes (around 8k to 16k environment steps) to recover the initial score, and that our method achieves best asymptotic scores both with SAC and CQL objectives.

Balanced replay analysis. We provide analysis for balanced replay in all manipulation tasks considered. As seen in Figure 17, without any privileged information, balanced replay automatically selects relevant offline samples for updates, while filtering out task-irrelevant, random data as fine-tuning proceeds.