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 $\mathcal{L}^{\texttt{CQL}}_{\texttt{critic}}(\theta_i)$ and update θ_i
    Calculate $\mathcal{L}^{\texttt{SAC}}_{\texttt{actor}}(\phi_i)$ 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 $\pi_\phi(\cdot|s) := \mathcal{N}\left(\frac{1}{N}\sum_{i=1}^{N}\mu_{\phi_i}(s), \quad \frac{1}{N}\sum_{i=1}^{N}(\sigma^2_{\phi_i}(s) + \mu^2_{\phi_i}(s)) - \mu^2_{\phi}(s)\right)$
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 $\mathcal{L}^{\texttt{SAC}}_{\texttt{critic}}(\theta)$ and update θ
  Calculate $\mathcal{L}^{\texttt{SAC}}_{\texttt{actor}}(\phi)$ 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

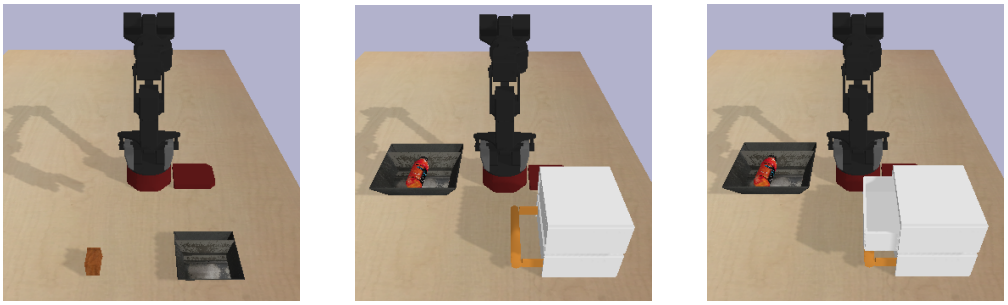

Figure 7: Manipulation tasks. (Left) `pick-place`, (Middle) `grasp-closed-drawer`, (Right) `grasp-blocked-drawer`.

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

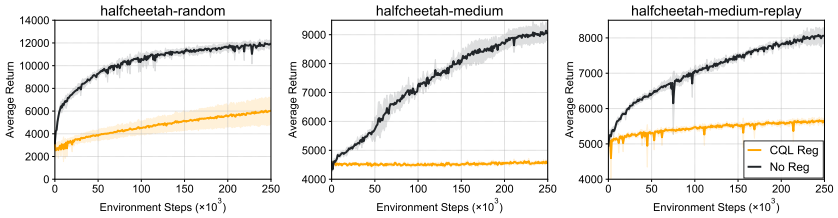

Figure 8: Fine-tuning performance with CQL regularization for `halfcheetah` tasks. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

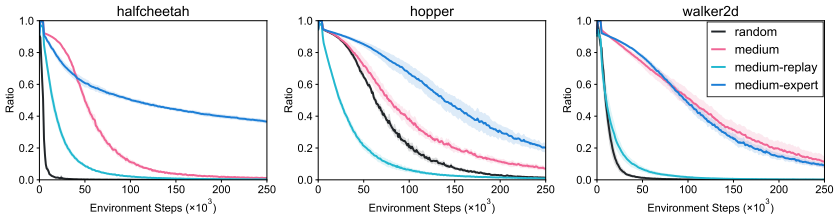

Figure 9: Proportion of offline samples used for updates during fine-tuning. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

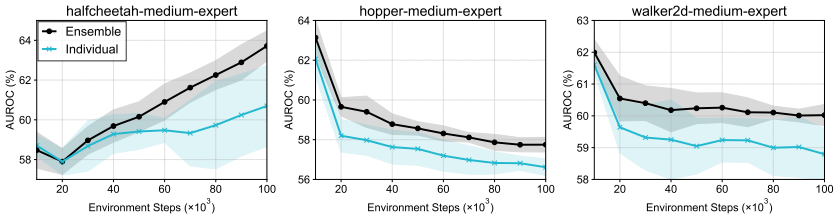

Figure 10: AUROC (%) over the course of fine-tuning on `medium-expert` tasks, where the Q-function is interpreted as a binary classifier that classifies a given state-action pair (s, a) as either a seen pair $(s, a_{\texttt{seen}})$ or an unseen pair $(s, a_{\texttt{uniform}})$, for a state s encountered 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. We remark that removing the regularization term may sometimes result in initial instability when working with narrowly distributed datasets, e.g., medium-expert, as seen in Figure 3. For applications that significantly favor safety over rapid fine-tuning, more conservative approaches such as deployment-efficient RL [47] may be more appropriate.

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 $\mathcal{D}^{\texttt{real}}_T := \{(s_i, a_i)\}_{i=1}^T$ be the samples collected online up until timestep T, we construct a corresponding fake dataset by replacing the actions in $\mathcal{D}^{\texttt{real}}_T$ by uniform random actions, i.e., $\mathcal{D}^{\texttt{fake}}_T := \{(s_i, a_{\texttt{unif}})\}_{i=1}^T$. 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.

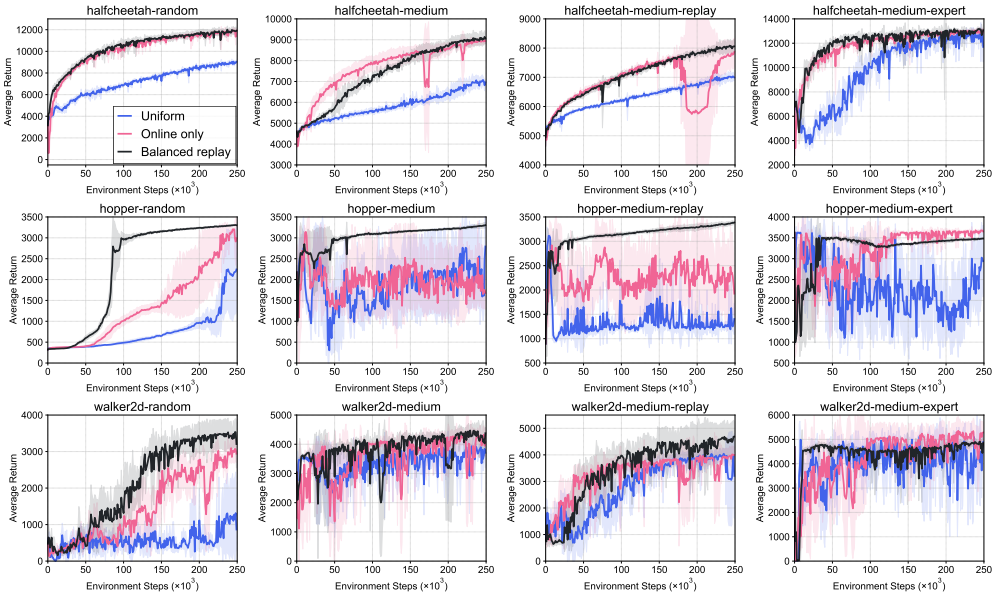

Figure 11: Ablation on balanced replay. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

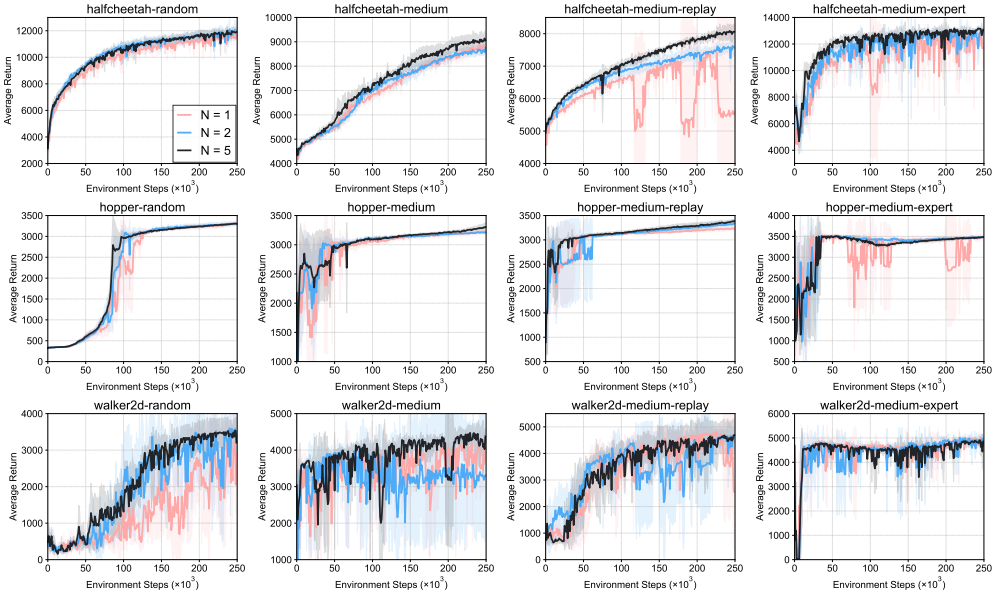

Figure 12: Fine-tuning performance with varying ensemble sizes. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

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.

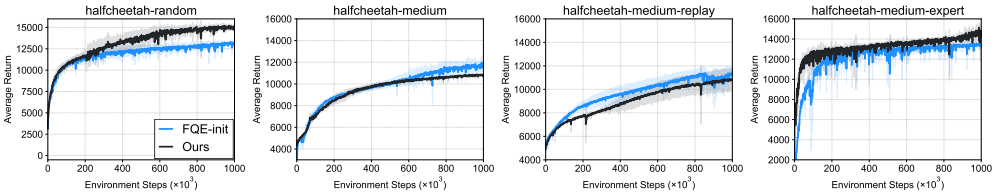

Figure 13: Asymptotic performance of our method (pessimistic Q-function) and FQE-init that fine-tunes a neutral Q-function. Ensemble and balanced replay were applied to both methods. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

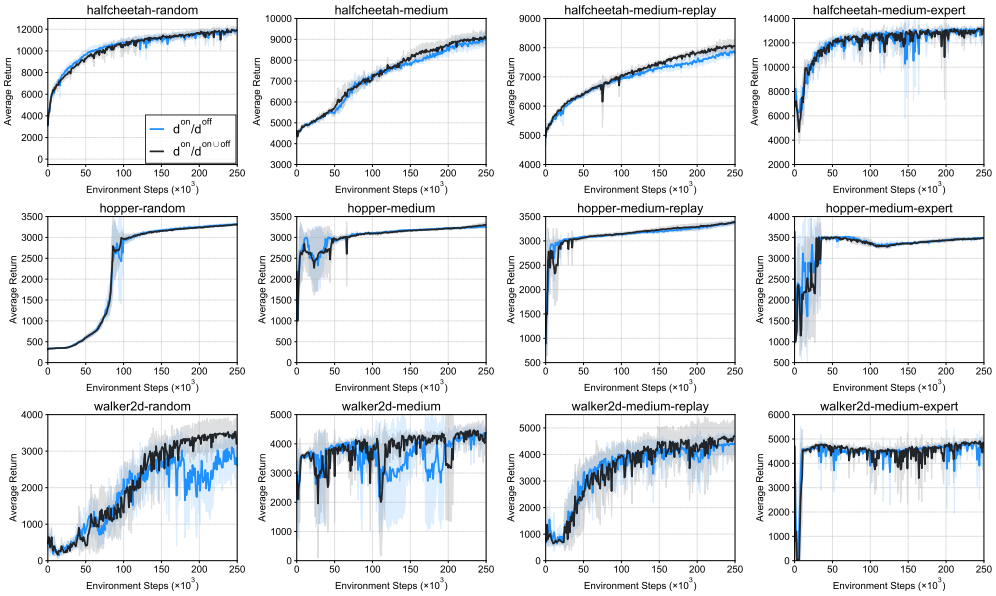

Figure 14: An alternative density ratio ($d^{\texttt{on}}/d^{\texttt{on}\cup\texttt{off}}$) for balanced replay. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

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.

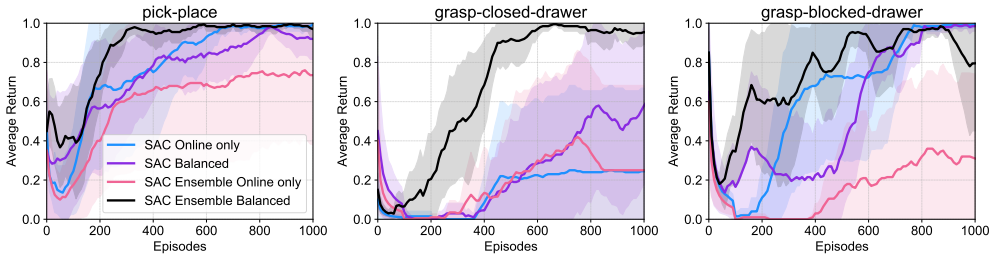

Figure 15: Ablation study with various ablated versions of our method, but with SAC objective. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

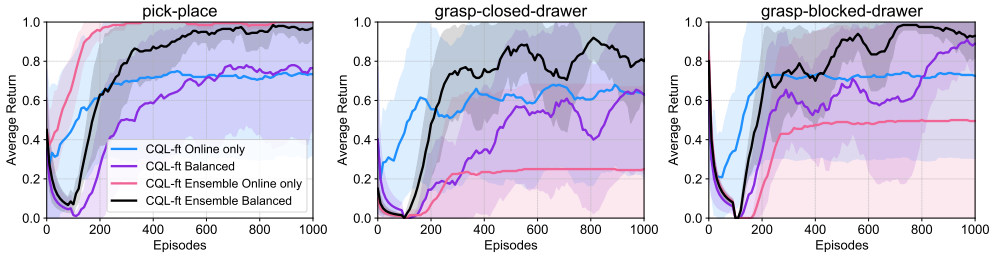

Figure 16: Ablation study with various ablated versions of our method, but with CQL objective. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

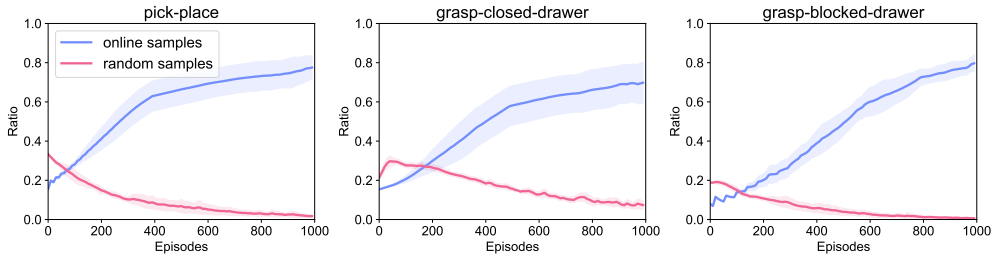

Figure 17: Proportion of random data used for during fine-tuning decreases over time. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

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.