COE: MITIGATING DEGRADATION WHILE IMPROVING FINE-TUNING IN OFFLINE-TO-ONLINE RL

Anonymous authors
Paper under double-blind review

ABSTRACT

In offline-to-online RL, agents often experience performance degradation in the early stages of fine-tuning. There is some evidence that offline (batch) RL algorithms can mitigate this degradation, but can also learn too slowly during fine-tuning. The majority of progress in offline-to-online RL has been on achieving faster fine-tuning, but with insufficiently controlled performance degradation. We propose a new algorithm, called Cautious Online Expansion (COE), that combines a conservative batch algorithm and a faster-learning online algorithm, using off-policy estimation to gradually hand off control. We show the resulting algorithm mitigates degradation while allowing for fast learning online, compared to several batch algorithms and offline-to-online algorithms. We provide theoretical support for the backstepping approach in the algorithm and how the policies are evaluated to effectively decide on the hand-off. We additionally ablate several design decisions in COE, providing empirical support for each component.

1 INTRODUCTION

Fine-tuning allows policies learned offline (Levine et al., 2020) to improve with additional interaction in the real environment (Nair et al., 2020; Lee et al., 2021), including on physical robots (Julian et al., 2020; Smith et al., 2022). Ideally, fine-tuning would improve the policy monotonically, but performance degradation is often observed in the early fine-tuning stage (Nair et al., 2020; Lee et al., 2021; Nakamoto et al., 2023). In low-risk scenarios, it may be acceptable for performance to degrade before it gets better, but, in many cases sharp performance degradation is unacceptable. Consider controlling fermentation in a brewery, where temperature must be held within a narrow range for the yeast to both reduce costs and produce good beer (Rajasekhar et al., 2024). Past batches provide ample offline data from existing temperature schedules, which are not optimal but reliably produce acceptable beer. Because no two batches are identical, the agent should keep adapting online (Petsagkourakis et al., 2020; Yoo et al., 2021), but a ruined batch is costly, so it must improve without too much degradation.

Various hypotheses have been explored in the literature to understand this performance degradation, including representation drift and forgetting (Campos et al., 2021; Wołczyk et al., 2024), the distribution shift from offline to online data (Lee et al., 2021; Zhao et al., 2022), and inaccurate action values on out-of-distribution actions that are amplified by bootstrapping (Lee et al., 2021; Nakamoto et al., 2023). Our results support this last issue, showing that out-of-distribution actions and states are a key culprit (see Appendix F).

Conservative offline (batch) RL algorithms are specifically designed to handle out-of-distribution issues. Conservative Q-learning (CQL) (Kumar et al., 2020) adds a regularizer that minimizes action-values outside of the dataset. Mildly conservative Q-learning (MCQ) (Lyu et al., 2022) instead trains the values of out-of-distribution actions toward pseudo-targets, a milder form of conservatism. PARS (Kim et al., 2025) penalizes values of infeasible actions to suppress extrapolation and incorporates a behavior cloning regularizer to stay closer to the dataset distribution. Implicit Q-learning (IQL) (Kostrikov et al., 2022) uses an expectile loss to greedify while only updating on actions in the dataset. In-sample Actor-Critic (InAC) (Xiao et al., 2023) limits the support of the learned policy by updating only on actions in the dataset. Such batch algorithms can naturally be used for the offline-to-online setting—and have been—because it is straightforward to first train on the offline

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



Figure 1: Cautious Online Expansion (COE) at a glance using the example of the HalfCheetah environment.

data and then continue updating online, initializing their buffers to the offline dataset and continuing with batch updating.

However, several works have highlighted that fine-tuning can be slow due to these offline methods being too conservative (Li et al., 2025; Kumar et al., 2020; Zhou et al., 2025). Cal-QL (Nakamoto et al., 2023) calibrates action-values for CQL to allow for faster fine-tuning, though ends up incurring some performance degradation. Warm-start RL (WSRL) (Zhou et al., 2025) discards old data as part of its approach, but ultimately also suffers from degradation, as does a related algorithm they compare to called Jump-Start RL (Uchendu et al., 2023). Policy Expansion (PEX) (Zhang et al., 2023) samples actions from a set of growing candidate policies, one of which is the offline learned policy. They show more consistent performance—both preventing degradation and learning efficiently—but only tested the approach on a subset of datasets. At the same time, the more recent offline PARS method demonstrated it performed well in fine-tuning. With the sheer volume of work in offline-online RL, most of which does not appropriately measure performance degradation, the current state of the field in terms of performance degradation and speed of learning online is unclear.[1]

We first revisit the use of batch (offline) RL algorithms for the offline-to-online RL setting, to assess if they do resolve degradation and allow performance improvement simultaneously. We find generally good performance, particularly from InAC and PARS, except for a few settings where indeed they do learn too slowly for fine-tuning. Such slow learning is not inherent, as the comparator baseline that uses SAC (Haarnoja et al., 2018) starting from a policy learned offline from InAC can perform well when used in the online phase, though it suffers from severe performance degradation. It remains open how to appropriately balance between the conservative batch algorithms, like InAC, to avoid degradation while benefitting from increased exploration from algorithms like SAC.

We propose an algorithm, called Cautious Online Expansion (COE), to combine the benefits of a conservative offline algorithm and a more exploratory online algorithm. The key idea behind COE is for the conservative agent (InAC) to initially take actions for the majority of the episode and only allow the exploratory agent (SAC) to take actions for a small number of steps at the end of the episode—which we call the *switch horizon*—slowly and cautiously expanding the steps the exploratory agent can control, as in Figure 1. We first provide theory supporting this approach. We then evaluate COE empirically, showing it improves on both degradation and fine-tuning performance compared to the batch algorithms and a variety of offline-to-online algorithms. We also ablate design choices in COE and find it is key to have the conservative policy continue to update (rather than be frozen), to dynamically adapt the switch horizon (increasing and decreasing) depending on online performance estimates, and additionally filter potentially low-value actions in the conservative policy.

2 Problem Setting and Background

In this paper we consider problems formulated as finite-horizon Markov Decision Processes (MDP) $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho, \gamma, H \rangle$. $\mathcal{S} \subseteq \mathbb{R}^{d_s}$ is the state space and $\mathcal{A} \subseteq \mathbb{R}^{d_a}$ is the action space, either

[1] An expanded related work section is in Appendix B.

of which can be discrete or continuous. The transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ describes the probability of transitioning from a state action pair to another state. The reward function is defined as $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, the discount factor is $\gamma \in [0, 1]$ and the maximum horizon for an episode is H. A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ induces the *return* from state s_t, $G_t \coloneqq r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{T-t-1} r_T$, which is the discounted sum of rewards from s_t until the end of the episode $T \leq H$. Under the initial-state distribution ρ, the policy performance is defined as $J(\pi) \coloneqq \mathbb{E}^{\pi}_{\rho}[G_0]$. Note we assume state can include the timestep t in the episode.

Offline-to-online learning consists of two phases: offline learning, followed by online fine-tuning. In the offline learning phase, a parameterized policy, $\pi_\phi : \mathcal{S} \to \Delta(\mathcal{A})$ is learned from a static dataset $\mathcal{D}_{\text{off}} = \{\langle s, a, s', r, \text{done}\rangle_i\}$, generated by some (potentially unknown) policy $\pi_{\mathcal{D}}$. During online fine-tuning, the agent interacts with the same MDP and updates its policy π_ϕ to improve performance. We assume a setting where the agent still has access to the offline data in the online phase, and can seed its online buffer with this offline data. The focus of this work is algorithms that improve online while limiting performance degradation, which we define in the next section.

All the methods we consider in this paper learn Q_θ, a parameterized action value function. $Q_\theta(s, a)$ estimates the expected return under π, $\mathbb{E}_\pi[G_t | S_t = s, A_t = a]$, starting from state-action pair (s, a) and taking actions according to π. More specifically, many of the algorithms we explore in this paper are based on the SAC update. SAC minimizes a KL-divergence to the Boltzmann policy $\pi(a|s) \propto \exp \frac{Q_\theta(s,a)}{\tau}$, where τ is the entropy parameter. A higher τ encourages higher entropy and more exploration. InAC is an offline learning algorithm with an update similar to SAC but designed to prevent bootstrapping from out-of-distribution actions (Xiao et al., 2023). InAC constrains the actor update by using a slightly different target policy in the KL-divergence

$$\pi(a|s) \propto \exp\left(\frac{Q_\theta(s,a)}{\tau}\right) = \frac{\pi_{\mathcal{D}}(a|s)}{\pi_{\mathcal{D}}(a|s)} \exp\left(\frac{Q_\theta(s,a)}{\tau}\right) = \pi_{\mathcal{D}}(a|s) \exp\left(\frac{Q_\theta(s,a)}{\tau} - \log \pi_{\mathcal{D}}(a|s)\right)$$

where $\pi(a|s) = 0$ if $\pi_{\mathcal{D}}(a|s) = 0$, which is the case for out-of-distribution actions. InAC does not require $\pi_{\mathcal{D}}$, it learns an estimate $\hat{\pi}_{\mathcal{D}}$ from the offline data and only uses it to compute this additional (regularization) term $-\log \hat{\pi}_{\mathcal{D}}(a|s)$ in the actor update.

3 Measuring Degradation and Improvement

In this section, we outline how to estimate performance degradation and performance improvement in the online phase.

Offline Policy Evaluation and Online Fine-Tuning Performance. Let π_{off} be the policy learned offline for some seed s. We run this fixed policy in the environment to generate n_{off} episodic returns $G^{\text{off}}_1[s], G^{\text{off}}_2[s], \ldots, G^{\text{off}}_{n_{\text{off}}}[s]$. The baseline performance, before starting online fine-tuning is defined as the average of these returns, $B[s] = \frac{1}{n_{\text{off}}} \sum_{i=1}^{n_{\text{off}}} G^{\text{off}}_i[s]$. For many of our experiments, $n = 30$, to give a reasonable sample average for this fixed policy.

The fine-tuning is evaluated according to online returns, generated while interacting with and learning in the environment. The agent learns for n episodes, yielding episodic returns $G_1[s], \ldots, G_n[s]$. For many of our experiments, $n = 500$, though the exact number of episodes can vary because the number of steps is fixed at 500k. We average over rolling windows of $w = 30$ episodes, to provide a less noisy estimate of the performance of the current agent.[2] This yields a new trajectory of online returns, $\bar{G}_1[s], \ldots, \bar{G}_T[s]$. This averaging smooths out one-off failures due to environment stochasticity while preserving sustained performance drops caused by policy degradation.[3] To plot the online performance vs the number of updates, we take the time step t and plot the $\bar{G}_i[s]$ that includes the return observed at timestep t.

[2] For the first 30 episodes, averaging is over a smaller window, using only returns available up to that point.

[3] A common evaluation in offline-to-online RL is to use evaluation rollouts (e.g., the scheme used for PARS (Kim et al., 2025)). This involves pausing training at regular intervals to compute means over multiple rollouts from a frozen (deterministic) policy. This scheme is simpler when plotting the number of updates on the x-axis of learning curves, but is not an accurate measure of how the fine-tuning policy will actually behave when deployed in the environment. Rather, it provides an overly optimistic view of its performance, because it reflects how good the deterministic policy could be if we got to start running it rather than the actual performance of the stochastic, learning policy.

Maximum Performance Degradation (MPD) measures the biggest drop in performance during online fine-tuning relative to the offline baseline. For seed s, we compute the difference of the minimum point from the online returns $\bar{G}_i[s]$ and the offline baseline $B[s]$:

$$\text{MPD}[s] := \max\left(B[s] - \min_{i\in\{1,\ldots,T\}} \bar{G}_i[s],\ 0\right) \tag{1}$$

We take the max with 0 for the rare case that the worst-case online performance is strictly greater than the baseline performance, due to learning quickly in the first few episodes and consistently staying above the baseline performance. This would result in a negative difference, implying performance improvement, but our goal here is to measure the level of degradation, which would be zero degradation in this case. This metric is analogous to the maximum drawdown used in financial time series, which measures the largest drop from a reference level during a trajectory.

We want to avoid performance degradation for most runs, rather than on average. A typical way to reflect this is to use an upper percentile over samples, such as the 90th percentile, wherein higher values indicate more severe performance degradation. Using this measure better reflects robustness across runs and is typically called the Value at Risk (VaR). We, therefore, use the VaR over the MPD across runs. Specifically, we compute the MPD per seed $s \in \mathcal{C}$, producing a set $\{\text{MPD}[s]\}_{s\in C}$ and measure

$$\text{VaR-MPD} = Q_{90}(\{\text{MPD}[s]\}_{s\in C}) \tag{2}$$

the 90-th percentile across the $|\mathcal{C}|$ MPD measures. Note that any percentile could be used: 95th for an even more conservative measure or 50th to report the median. We opt for the 90th percentile as a typical choice for VaR.

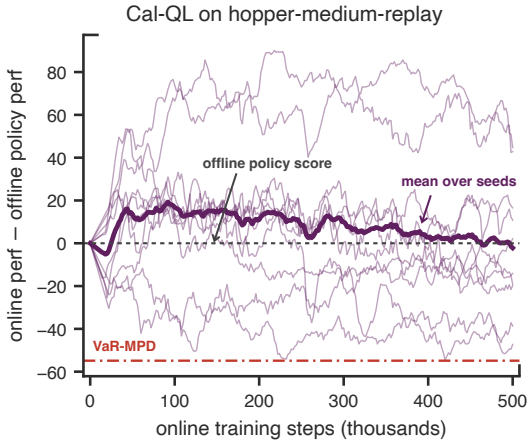


Figure 2: **Evaluating using mean returns can hide per-seed performance degradation.** Cal-QL is fine-tuned online for 500k training steps on hopper-medium-replay. Each thin line is one seed's online score minus its own offline baseline B. The offline baseline B is the 30-episode stochastic rollout score of the offline checkpoint that the seed starts from. Online scores are the rolling mean over the last 30 training episodes, and every curve starts at B. The mean score over seeds, indicated by the thicker line, peaks at $+19$ near 90k steps and ends near -2, which looks like a very mild amount of degradation. However, the individual seeds fluctuate wildly, with majority of seeds going below the baseline B at some point. Var-MPD, indicated by the red line, correctly measures the performance degradation at 55 points.

Performance Gain (PG) measures the extent to which the online fine-tuning results in a policy which outperforms the offline policy given at initialization. The PG is the improvement over the offline baseline $B[s]$, according to $\bar{G}_T[s]$, which is the mean of last $w = 30$ online returns:

$$\text{PG}[s] := \max\left(\bar{G}_T[s] - B[s],\ 0\right) \tag{3}$$

Final Performance (FP) is similar to PG, but omits the baseline: $\text{FP}[s] := \bar{G}_T[s]$.

In our experiments, we report the average improvement over seeds, Mean-PG $:= \frac{1}{|\mathcal{C}|}\sum_{s\in\mathcal{C}} \text{PG}[s]$. We use mean rather than VaR because our goal with final performance is to understand overall improvement from the algorithm. We want to ensure that the agent avoids performance drop in most runs, because we want to avoid performance drop robustly across different locations the agent is deployed. But overall performance gains do not need to be robust in the same way, and an average more simply indicates overall performance improvements across locations the agent is deployed.

4 Can we just use offline (batch) RL algorithms?

In this section we investigate the performance degradation and gain for four batch RL algorithms: PARS, InAC, IQL and CQL. We chose these four offline algorithms because they are the basis

for several offline-to-online algorithms and because they all aim to avoid bootstrapping on out-of-distribution actions. We chose PARS over ReBRAC (Tarasov et al., 2023), as the PARS paper showed improved performance over ReBRAC. We also use a baseline that is likely to suffer from degradation but potentially allow for faster learning online: learning with SAC online after initializing with the InAC policy learned offline. Note that for all algorithms we incorporated several normalizations that were key for more stable performance across settings, including observation and reward normalization as well as using layernorm. We briefly support the importance of this modification in Appendix 7.

Experimental Settings: We tested these algorithms on 16 D4RL datasets (see Table 1 for the complete list). All agents learn online for 5×10^5 environment steps. The scores are normalized in D4RL, where a score of 0 corresponds to a random policy while a score of 100 indicates that the performance matches that of D4RL expert reference for that task. Each experiment is repeated for 15 seeds. When computing the VaR-MPD for an environment, the VaR is taken over these 15 seeds; for mean-FP, the average is taken over these 15 seeds. For some plots, the results are also aggregated over all 16 environment-dataset settings, where we take the mean of VaR-MPD or mean of mean-FP over these 16.

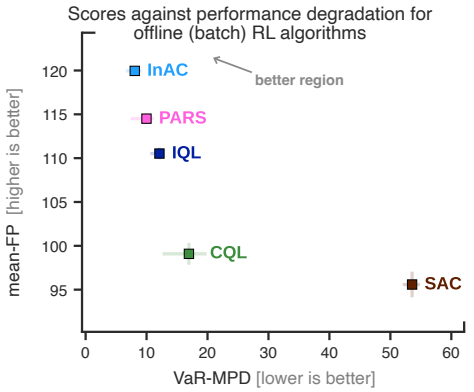


Figure 3: **Mean-FP versus VaR-MPD for offline RL batch algorithms.** Top-left is the best region. Bars indicate 95% stratified bootstrap confidence intervals. The VaR-MPD intervals are asymmetric because no resample can produce a 90th percentile above the worst seed's value regardless of the number of random draws.

All networks are two layer MLPs of width 256 units. Optimization is done using Adam at 3×10^{-4} (CQL actor uses 10^{-4}), $\gamma = 0.99$, Polyak rate 0.005 and batches of 256. It is common practice to tune hyperparameters per environment, using performance in deployment. For example, PARS had different hyperparameters per environment in their work, obtained using online performance. However, we do not adopt this unrealistic practice here; in the real world, they cannot be tuned using sweeps in deployment. We aimed for minimal tuning, setting to reasonable defaults for most hyperparmeters based on performance in the original work and only tuning one or two hyperparameters that were particularly sensitive. This tuning was done across environments (using only a random subset of 5 environments) and then fixed for all experiments. This tuning, the resulting hyperparameters and other experimental details are described in more detail in Appendix C.

Outcomes: We see in Figure 3 that the batch algorithms can perform well, achieving above 100 in final performance and significantly reducing degradation over SAC. InAC performs the best in terms of both final performance and performance degradation, followed by PARS, then IQL and finally CQL performs notably worse. We would expect SAC—which corresponds to training offline first with InAC and then continuing online with the SAC update—to have performance degradation, but might expect higher final performance. We see worse final performance because for some settings it was not able to recover from the degradation within the fine-tuning window. When it can recover, it can get to good or even notably better final performance (see Appendix E.3 for learning curves).

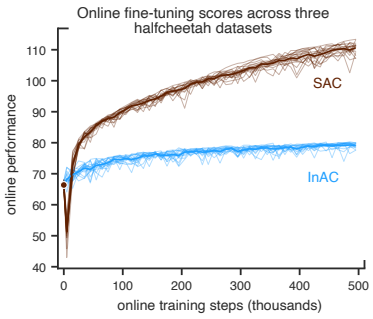


Figure 4: **Learning curves** for InAC and SAC in HalfCheetah, with the plot including all individual learning curves and a darker line as the average curve across 15 runs and three datasets: medium-replay, medium-expert, and expert.

We show one such setting, where SAC is able to get to good final performance, but the batch algorithms struggled. Figure 4 depicts the learning curve in HalfCheetah, across three different datasets, for SAC and InAC. Note the behavior for the other batch algorithms was similar to InAC, and so we only visualize one. InAC is unable to noticeably improve over its

offline policy. SAC degrades in a steep drop early on, but then quickly recovers and surpasses a score of 100.

5 The Cautious Online Expansion (COE) Algorithm

Our goal is to design an algorithm that largely mitigates performance degradation, like an offline learning algorithm, but learns more quickly during fine-tuning, like SAC. We design the Cautious Online Expansion (COE) algorithm to do just that, by automatically deciding when to expand exploration versus using a more conservative policy update during online fine-tuning. COE maintains two agents: π^c that uses more conservative updates and π^e that uses more aggressive exploratory policy updates. It also maintains a switch horizon $h \in [0, H]$, where H is the maximum episode length. COE takes actions according to the composite policy

$$\pi^h(\pi^c, \pi^e, t)(\cdot \mid s) = \begin{cases} \pi^c(\cdot \mid s), & t < h, \\ \pi^e(\cdot \mid s), & t \geq h. \end{cases}$$

Decreasing h extends the explorer's control to earlier in the episode, while increasing h gives control to the conservative policy. The conservative policy π^c is updated online using the offline RL algorithm InAC. The exploratory policy π^e uses SAC, which generates more novel data for both π^c and π^e to learn from. We provide theoretical support for stepping backwards in the next section.

COE dynamically decreases or increases h by evaluating the policies using Fitted Q Evaluation (FQE) (Le et al., 2019) and sampled returns. We choose FQE as our policy evaluation algorithm because of its simplicity and its ability to scale to continuous and large state–action spaces. Particularly useful for us is that we get (nearly) on-policy samples of our current policy, removing some of the hardness issues with off-policy policy evaluation (Wang et al., 2021). FQE learns an estimate $\hat{Q}^\pi$ of the action-value for a fixed policy π using the same dataset/buffer of tuples used for the critics. The performance estimate is computed as the sample average over all start states $s \in \mathcal{D}_0$ in the dataset and n actions sampled to approximate the expectation $\mathbb{E}_\pi[\hat{Q}^\pi(s_0, A)]$

$$\hat{v}(\pi, \hat{Q}^\pi, \mathcal{D}_0) \coloneqq \frac{1}{|\mathcal{D}_0| n} \sum_{s_0 \in \mathcal{D}_0} \sum_{a_1, \ldots, a_n \sim \pi(\cdot|s_0)} \hat{Q}^\pi(s_0, a_i) \tag{4}$$

COE has separate FQE critics for the composite and conservative policies, denoted $\hat{Q}^{\pi^h}$ and $\hat{Q}^{\pi^c}$.

The horizon h is increased or decreased as follows. The switch horizon h is decreased if $\hat{v}(\pi^h, \hat{Q}^{\pi^h}, \mathcal{D}_0) > \hat{v}(\pi^c, \hat{Q}^{\pi^c}, \mathcal{D}_0)$, i.e., less caution if the composite policy has improved over the conservative policy. COE also tracks if performance has degraded once the horizon is decreased. To be more robust to bias in the FQE estimates, the switch horizon is increased based on sampled returns. Specifically, h is increased if performance has degraded over the last few episodes, according to the average of the returns over those episodes.

There are also two important modifications we made to InAC, particularly to make it more conservative. With an exploratory component in COE, we found we could make the conservative component more conservative, further reducing degradation while still allowing for performance gains. First, we added behavior cloning to InAC, both offline and online. Second, we screened low-value actions sampled from π^c. Specifically, when we sample $a \sim \pi^c(\cdot|s)$, if the value $Q(s, a) < Q(s, \mu)$ for the mode action μ of $\pi^c(\cdot|s)$, then we instead take action μ. Since we use Gaussians, the mean and mode are equivalent. This *mode-screening* avoids low probability but potentially bad actions from being taken by the conservative policy, further limiting how much it can explore.

We summarize COE in Algorithm 1 in Appendix D, and also provide other details including how the loss is changed for InAC to incorporate behavior cloning and default hyperparameter settings.

Remark: There are two other methods that switch between policies in the online phase: JSRL (Uchendu et al., 2023) and PEX (Zhang et al., 2023). JSRL was introduced as a framework to promote exploration, using a frozen guide policy to create a curriculum for the learning policy. It was applied to many settings, including the offline-to-online setting. It similarly steps back from the end of the episode, but was not designed to mitigate performance degradation, as we show in our experiments. PEX uses a mechanism similar to our screening method above. It keeps a set of candidate policies, including the offline policy. On each step, it samples an action from each candidate and

then uses a softmax (Boltzmann) weighting on the critic values for those actions to probabilistically decide which action to execute. Their goal was to avoid overwriting the offline policy, while still allowing learning online, but the mechanism is not suitable for avoiding degradation.

6 Theoretical Results

In this section we provide theoretical support for COE. Theorem 6.1 shows that some risk of degradation is generally unavoidable. Thus COE does not aim for zero degradation, but to balance degradation control and continued improvement. Theorem 6.2 motivates stepping the switch horizon backwards from the end of the episode. Proposition 1 motivates that using sampled returns can recognize degradation in only a few episodes, allowing COE to increase the switch horizon and limit persistent degradation.

Let $\mathcal{M}$ denote the class of finite-horizon MDPs considered in Section 2 with $0 < \gamma < 1$, rewards bounded in magnitude by $R_{\max} > 0$, and an action space containing at least two actions. For $M \in \mathcal{M}$, let $J_M(\pi)$ be the performance of policy π in M. Let Π_M be the class of possibly time-dependent stochastic Markov policies for M. Let $\pi^{(0)}$ be the offline policy—and the initial policy during fine-tuning—with a sequence of policies $\pi^{(1)}, \pi^{(2)}, \ldots$ produced after each update online. We assume randomized learners that depend only on the offline dataset, observed online interaction, and internal randomness. Proofs for all theoretical results are provided in Appendix A.

We first establish that zero degradation and nontrivial performance improvement cannot in general be achieved simultaneously. Probabilities below are conditional on the offline dataset $\mathcal{D}_{\text{off}}$ and $\pi^{(0)}$.

Theorem 6.1 (Impossibility of universal zero degradation)**.** *Fix $K \geq 1$. No offline-to-online learning algorithm can, for every $M \in \mathcal{M}$ and every finite offline dataset that can arise in M, simultaneously satisfy:*

(i) ***Zero degradation:*** $\mathbb{P}_M\big(J_M(\pi^{(k)}) \geq J_M(\pi^{(0)})\ \forall k = 1, \ldots, K\big) = 1$*;*

(ii) ***Nontrivial improvement:*** *whenever there is a policy $\bar{\pi} \in \Pi_M$ where $J_M(\bar{\pi}) > J_M(\pi^{(0)})$, the algorithm satisfies $\mathbb{P}_M\big(J_M(\pi^{(k)}) > J_M(\pi^{(0)})$ for some $k = 1, \ldots, K\big) > 0$.*

We next show that COE's backwards expansion has a useful worst-case property. For $\mathcal{I} \subseteq \{0, \ldots, H-1\}$, let $\pi_{\mathcal{I}}$ use π^e at time steps in $\mathcal{I}$ and π^c otherwise. For nonempty $\mathcal{I}$, define $\mathcal{R}(\mathcal{I}) := \sup_{M \in \mathcal{M}, \pi^c, \pi^e \in \Pi_M} \left[J_M(\pi^c) - J_M(\pi_{\mathcal{I}})\right]_+$.

Theorem 6.2 (Backwards expansion minimizes worst-case degradation)**.** *For every $1 \leq d \leq H$,*

$$\operatorname*{arg\,min}_{\mathcal{I} \subseteq \{0,\ldots,H-1\}, |\mathcal{I}| = d} \mathcal{R}(\mathcal{I}) = \{\mathcal{I}_d^\star\}, \qquad \textit{where } \mathcal{I}_d^\star = \{H-d, \ldots, H-1\}.$$

Therefore, for a fixed amount of explorer exposure (d steps in the episode), placing that exposure at the end of the episode uniquely minimizes worst-case degradation relative to the conservative policy. This motivates why COE expands explorer control backwards by decreasing h.

Expansion enables the explorer to collect experience at new time steps, but can also reduce realized performance. COE therefore couples expansion with a rollback mechanism that retracts explorer control following sufficiently large sustained performance drops. The next result in Proposition 1 characterizes when rollback is triggered. The larger the sustained gap below the rollback threshold, the less likely the expansion is to remain deployed for the full five-episode window.

Proposition 1 (Finite-time rollback under persistent degradation)**.** *Let c be a rollback threshold fixed by the pre-expansion history. Suppose episodic returns lie in an interval of width $B > 0$. If, while the expansion remains deployed, the conditional expected return of each post-expansion episode is at least $\varepsilon > 0$ below c. Then a rollback test using the mean of m post-expansion returns satisfies, conditional on the pre-expansion history,*

$$\mathbb{P}(\textit{no rollback within } m \textit{ episodes}) \leq \exp\left(-\frac{2m\varepsilon^2}{B^2}\right).$$

COE uses $m = 5$ and $c = \bar{Z} - \sigma/\sqrt{30}$, giving $\mathbb{P}(\text{no rollback within 5 episodes}) \leq \exp\left(-\frac{10\varepsilon^2}{B^2}\right)$.

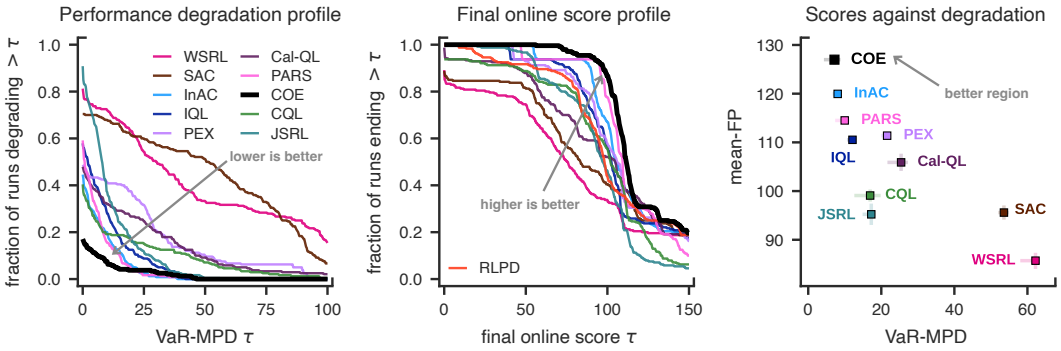


Figure 5: **Performance degradation and online scores over 16 environments and 15 seeds. Left:** the fraction of runs (out of 240) whose performances degrade more than τ relative to their own offline policy baseline score B; lower is better. **Middle:** the fraction of runs whose final online scores are greater than τ; higher is better. **Right:** Means of Final Scores versus VaR-MPD, where top-left region is the best. Bars indicate 95% stratified bootstrap confidence intervals. RLPD trains from scratch with no offline policy, so it appears only in the middle figure. All these results indicate that COE is able to achieve the lowest performance degradation and highest final online scores.

7 Comparison to methods improving fine-tuning

In this section we expand our empirical study to several offline-to-online methods, including RLPD, WSRL, JSRL, PEX and Cal-QL. We evaluate under the same experimental protocol as in Section 4, across the same 16 D4RL datasets. We include additional details on implementation choices and hyperparameters for these algorithms in Appendix C.

Across 16 environments, COE achieves a mean final online score of 127.7 ± 1.2, significantly above InAC's at 119.4 ± 0.3, which is the next best, with individual environment performance in Table 1. Its VaR-MPD is 7.2 ± 2.3, against 8.1 ± 1.3 for InAC and 10.0 ± 1.9 for PARS. InAC and PARS are both methods that carry over the pessimistic offline objective in the online fine-tuning phase. Since the confidence intervals overlap, it indicates that COE is at least as stable as the most conservative methods, while outperforming them in terms of final scores. COE is the only method that sits on the top-left corner in Fig. 5 (right). The other end of the region are occupied by methods that fine-tune the offline policy without any conservatism: WSRL and SAC. WSRL's policy degrades by 62.2 ± 2.5 and SAC's by 53.6 ± 1.4, and both of them converge to a solution worse than the starting offline policy on adroit manipulation tasks.

Performance profiles in Fig. 5 (left and middle) show that COE dominates all the other methods in terms of final performance and VaR-MPD across the vast majority of its seeds. Furthermore, when looking at the learning curves themselves, we also find that COE is able to learn faster than other baselines; we include this result in Figure 9 in Appendix E.

8 COE Component Ablations

We ablate COE by removing individual components along two axes: 1) changes to the conservative policy and 2) changes to the switch horizon. We evaluate these variants of COE on 6 environments using 5 seeds each, and analyze their performance degradations and final performance.

Conservative policy We compare COE against three variants of its conservative policy, labelled in blue in Figure 6: 1) the proposed COE, 2) COE without behavior cloning penalty in the InAC loss, 3) COE without *mode-screening* when sampling actions from the conservative policy, and 4) COE where the conservative policy is frozen during online fine-tuning and only the explorer policy is updated. Removing any of these components results in both a lower Mean-FP and higher VaR-MPD. Note that we also evaluated the conservative and exploratory policies for COE during learning, and find that if they were independently run, the agent would suffer degradation or slow learning, but the composite policy with a dynamic switch horizon gets the best of both (see Figure 10).

Table 1: **Per-environment normalized online scores using 15 seeds.** These scores are obtained by taking the online stochastic policy at the final training step and evaluating for 100 episodes. Subscript indicate 95% seed bootstrap.

task	RLPD	WSRL	SAC	InAC	IQL	PEX	Cal-QL	PARS	CQL	JSRL	COE
hc-mr	$82.0_{\pm 0.8}$	$88.0_{\pm 1.4}$	$98.5_{\pm 2.0}$	$55.1_{\pm 0.6}$	$39.8_{\pm 0.3}$	$42.3_{\pm 0.1}$	$81.8_{\pm 0.5}$	$47.4_{\pm 0.2}$	$83.0_{\pm 0.7}$	$56.9_{\pm 0.5}$	$98.0_{\pm 1.2}$
hc-m-exp	$98.1_{\pm 1.0}$	$104.4_{\pm 2.5}$	$115.3_{\pm 0.8}$	$90.2_{\pm 0.4}$	$81.0_{\pm 0.6}$	$91.9_{\pm 0.2}$	$94.9_{\pm 0.2}$	$95.4_{\pm 0.2}$	$101.5_{\pm 0.4}$	$88.1_{\pm 1.6}$	$112.1_{\pm 0.9}$
hc-exp	$98.2_{\pm 0.5}$	$108.4_{\pm 1.1}$	$118.3_{\pm 0.8}$	$92.3_{\pm 0.5}$	$85.8_{\pm 0.8}$	$94.4_{\pm 0.3}$	$109.5_{\pm 0.6}$	$98.4_{\pm 0.1}$	$110.7_{\pm 0.4}$	$92.8_{\pm 0.4}$	$115.5_{\pm 0.8}$
hopper-mr	$84.7_{\pm 8.8}$	$69.8_{\pm 11.7}$	$58.5_{\pm 2.9}$	$102.2_{\pm 0.6}$	$75.3_{\pm 2.6}$	$64.2_{\pm 5.3}$	$58.1_{\pm 4.5}$	$99.0_{\pm 4.6}$	$62.2_{\pm 7.8}$	$50.0_{\pm 4.0}$	$98.5_{\pm 6.8}$
hopper-m-exp	$90.2_{\pm 4.6}$	$56.9_{\pm 6.3}$	$63.2_{\pm 6.7}$	$106.4_{\pm 1.2}$	$101.7_{\pm 0.5}$	$108.6_{\pm 1.6}$	$74.5_{\pm 4.4}$	$108.8_{\pm 1.3}$	$106.9_{\pm 1.5}$	$99.3_{\pm 2.7}$	$101.2_{\pm 7.8}$
hopper-exp	$80.6_{\pm 8.5}$	$63.2_{\pm 6.8}$	$68.9_{\pm 9.9}$	$94.7_{\pm 2.9}$	$106.1_{\pm 0.8}$	$94.6_{\pm 3.4}$	$56.3_{\pm 2.3}$	$104.2_{\pm 1.9}$	$105.1_{\pm 2.0}$	$90.3_{\pm 2.9}$	$110.1_{\pm 1.8}$
walker-mr	$103.1_{\pm 3.4}$	$59.9_{\pm 13.1}$	$68.4_{\pm 15.7}$	$104.4_{\pm 1.1}$	$89.8_{\pm 0.7}$	$80.7_{\pm 1.0}$	$80.9_{\pm 14.1}$	$100.1_{\pm 1.8}$	$123.4_{\pm 2.6}$	$95.4_{\pm 4.7}$	$107.0_{\pm 8.0}$
walker-m-exp	$119.1_{\pm 2.2}$	$67.3_{\pm 9.3}$	$85.3_{\pm 20.0}$	$109.8_{\pm 0.3}$	$106.5_{\pm 0.4}$	$108.7_{\pm 0.2}$	$119.8_{\pm 4.0}$	$109.4_{\pm 0.1}$	$112.3_{\pm 0.3}$	$109.3_{\pm 0.2}$	$108.2_{\pm 14.1}$
walker-exp	$105.6_{\pm 14.3}$	$82.8_{\pm 5.4}$	$69.7_{\pm 13.1}$	$115.8_{\pm 1.2}$	$107.7_{\pm 0.2}$	$108.4_{\pm 0.2}$	$103.3_{\pm 13.4}$	$110.3_{\pm 0.1}$	$123.8_{\pm 0.6}$	$113.4_{\pm 0.5}$	$112.9_{\pm 3.3}$
maze2d-u	$167.1_{\pm 1.0}$	$167.9_{\pm 0.8}$	$167.3_{\pm 1.1}$	$165.8_{\pm 0.6}$	$154.0_{\pm 1.4}$	$149.1_{\pm 1.8}$	$165.4_{\pm 1.3}$	$129.0_{\pm 1.7}$	$81.4_{\pm 1.1}$	$65.0_{\pm 13.6}$	$162.0_{\pm 0.6}$
maze2d	$187.9_{\pm 0.6}$	$187.8_{\pm 0.8}$	$187.5_{\pm 0.7}$	$186.5_{\pm 0.7}$	$169.8_{\pm 1.8}$	$169.7_{\pm 2.1}$	$186.8_{\pm 0.7}$	$145.5_{\pm 1.7}$	$80.0_{\pm 7.0}$	$126.5_{\pm 18.6}$	$187.1_{\pm 0.8}$
maze2d-l	$246.5_{\pm 0.9}$	$242.5_{\pm 1.8}$	$240.4_{\pm 10.6}$	$242.1_{\pm 1.4}$	$205.6_{\pm 3.0}$	$198.3_{\pm 18.3}$	$242.2_{\pm 1.0}$	$171.6_{\pm 10.8}$	$213.9_{\pm 2.8}$	$104.9_{\pm 25.4}$	$243.4_{\pm 2.2}$
pen-exp	$58.2_{\pm 12.3}$	$-0.0_{\pm 1.3}$	$-1.9_{\pm 1.1}$	$131.9_{\pm 2.5}$	$135.4_{\pm 2.0}$	$145.0_{\pm 2.2}$	$84.9_{\pm 20.9}$	$148.3_{\pm 2.1}$	$53.2_{\pm 14.3}$	$117.3_{\pm 4.9}$	$145.3_{\pm 1.0}$
reloc-exp	$23.3_{\pm 4.7}$	$-0.2_{\pm 0.0}$	$-0.2_{\pm 0.0}$	$93.9_{\pm 1.5}$	$98.4_{\pm 0.9}$	$103.5_{\pm 1.0}$	$0.2_{\pm 0.1}$	$106.0_{\pm 1.2}$	$-0.1_{\pm 0.0}$	$94.0_{\pm 1.6}$	$107.3_{\pm 0.5}$
door-exp	$91.0_{\pm 3.3}$	$4.7_{\pm 6.6}$	$36.1_{\pm 18.3}$	$100.2_{\pm 0.7}$	$96.6_{\pm 0.8}$	$94.0_{\pm 4.2}$	$104.7_{\pm 0.7}$	$106.2_{\pm 0.1}$	$94.9_{\pm 3.4}$	$100.6_{\pm 0.7}$	$104.9_{\pm 0.5}$
hammer-exp	$134.0_{\pm 14.9}$	$88.9_{\pm 20.8}$	$137.5_{\pm 4.3}$	$119.7_{\pm 1.3}$	$107.2_{\pm 2.8}$	$120.9_{\pm 2.6}$	$132.8_{\pm 0.8}$	$129.5_{\pm 1.1}$	$134.9_{\pm 3.5}$	$115.5_{\pm 2.3}$	$129.5_{\pm 1.6}$
mean	$110.6_{\pm 1.9}$	$87.0_{\pm 2.0}$	$94.5_{\pm 2.4}$	$119.4_{\pm 0.3}$	$110.0_{\pm 0.4}$	$110.9_{\pm 1.4}$	$106.0_{\pm 1.8}$	$113.1_{\pm 0.8}$	$99.2_{\pm 1.2}$	$95.0_{\pm 2.2}$	$\mathbf{127.7}_{\pm 1.2}$

Switch horizon h In the proposed COE algorithm, the switch horizon h can be decreased using FQE evaluation of the composite and conservative policies, and increased if the return degrades over time. We test whether it is necessary to allow the switch horizon to increase, depicted in red in Figure 6, and see that it is critical. COE achieves a final online score of 140.2 ± 1.2 and VaR-MPD of 3.2 ± 2.0. Preventing the switch horizon from increasing when performance degrades reduces the online score to 129.9 ± 3.4 and increases VaR-MPD to 28.1 ± 2.9. We also investigated the importance of using FQE to decrease switch horizon h, instead decreasing h on a fixed schedule such that $h = 0$ at 2.5×10^5 online training steps (total online training budget is 5×10^5). This ablation results in an even lower score of 103.1 ± 2.5 and higher VaR-MPD of 55.4 ± 4.8.

We include additional results on hyperparameter sensitivity in Appendix E.2.

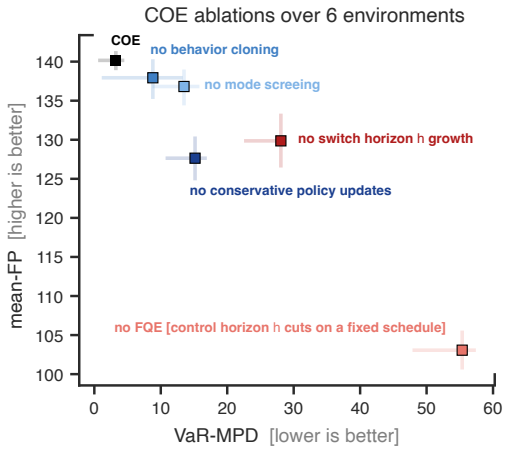


Figure 6: **Ablations comparing Mean-FP versus VaR-MPD for ablations of COE.** Each square represents a variant with one component removed from COE. The variants in blues are changes to the conservative policy algorithm, while the ones in reds are changes to how the switch horizon is chosen. The Mean-FP and VaR-MPD are computed using 5 seeds and 6 environments: {halfcheetah-medium-replay, hopper-medium-expert, walker2d-expert, maze2d-large, pen-expert, hammer-expert}. Bars indicate the stratified 95% bootstrap confidence intervals.

9 Conclusions

In this paper, we introduced the Cautious Online Expansion (COE) algorithm to both mitigate performance degradation while promoting faster online fine-tuning in the offline-to-online setting. We showed that offline batch algorithms, including InAC, PARS and IQL, are already quite effective, both largely preventing performance degradation and also often obtaining high final performance. But we found in a few cases they can learn slowly or almost not at all online. Other algorithms designed to learn faster online, such as JSRL, WSRL, RLPD, PEX and Cal-QL, can suffer significantly from performance degradation, which can also limit final performance. COE instead provides a better balance, starting with the already effective batch algorithm InAC and slowly handing off control to a more exploratory policy to allow for faster fine-tuning.

AI use statement

In this work, we used generative AI tools to formulate mathematical claims, particularly using it for brainstorming, to help type out proofs in latex and to verify proofs. We have not used generative AI

tools to generate synthetic data sets, design or provide feedback on research methodology or experiments, implement methods, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis nor interpret results.

We used generative AI tools to create images (specifically Fig. 1), find additional related work, list any errors in the paper (with no direct editing), grammar check, and reword some sentences

We used generative AI in a limited way for the code for experiments. The use with code was limited to adapting existing scripts written by the authors, such as modifying variables, changing data-fetch paths, refactoring, and providing debugging assistance.

We have reviewed all AI-assisted work. We reviewed any code that was used in the experiments, feasible due to the limited scope. We only used generative AI to find additional literature, in case we missed any, but did the literature survey ourselves. We directly edited the overview figure after it was generated, to ensure correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Reproducibility statement

We provide all the hyperparameters and technical details to help reproduce our work in Appendix C. Additionally, we uploaded an anonymous version of the codebase as supplementary material (on OpenReview) along with this submission, which includes the implementation of our proposed method, all the baselines used in our work as well as the scripts for computing the performance metrics proposed in the main text. After the review period, a public GitHub link will be provided here.

References

Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, 2023.

Victor Campos, Pablo Sprechmann, Steven Hansen, Andre Barreto, Steven Kapturowski, Alex Vitvitskyi, Adria Puigdomenech Badia, and Charles Blundell. Beyond fine-tuning: Transferring behavior in reinforcement learning. *arXiv preprint arXiv:2102.13515*, 2021.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *Advances in neural information processing systems*, 2021.

Nandiraju Gireesh, Yuanliang Ju, and He Wang. Adaptive q-chunking for offline-to-online reinforcement learning. *arXiv preprint arXiv:2605.05544*, 2026.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 2018.

Ryan Julian, Benjamin Swanson, Gaurav Sukhatme, Sergey Levine, Chelsea Finn, and Karol Hausman. Never stop learning: The effectiveness of fine-tuning in robotic reinforcement learning. In *Proceedings of Machine Learning Research*, 2020.

Jeonghye Kim, Yongjae Shin, Whiyoung Jung, Sunghoon Hong, Deunsol Yoon, Youngchul Sung, Kanghoon Lee, and Woohyung Lim. Penalizing infeasible actions and reward scaling in reinforcement learning with offline data. In *International Conference on Machine Learning*, 2025.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

Romain Laroche, Paul Trichelair, and Remi Tachet Des Combes. Safe policy improvement with baseline bootstrapping. In *International conference on machine learning*, pp. 3652–3661. PMLR, 2019.

Hoang Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, 2019.

Seunghyun Lee, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin. Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble. In *Annual Conference on Robot Learning*, 2021.

Sergey Levine, Aviral Kumar, George Tucker, Justin Fu, et al. Offline reinforcement learning: Tutorial, review. *and Perspectives on Open Problems*, 5, 2020.

Jianxiong Li, Xiao Hu, Haoran Xu, Jingjing Liu, Xianyuan Zhan, and Ya-Qin Zhang. PROTO: Iterative policy regularized offline-to-online reinforcement learning. *arXiv preprint arXiv:2305.15669*, 2023.

Lu Li, Tianwei Ni, Yihao Sun, and Pierre-Luc Bacon. The three regimes of offline-to-online reinforcement learning. *arXiv preprint arXiv:2510.01460*, 2025.

Qiyang Li, Zhiyuan Paul Zhou, and Sergey Levine. Reinforcement learning with action chunking. *Advances in Neural Information Processing Systems*, 38:55518–55553, 2026.

Yicheng Luo, Jackie Kay, Edward Grefenstette, and Marc Peter Deisenroth. Finetuning from offline reinforcement learning: Challenges, trade-offs and practical solutions. *arXiv preprint arXiv:2303.17396*, 2023.

Jiafei Lyu, Xiaoteng Ma, Xiu Li, and Zongqing Lu. Mildly conservative q-learning for offline reinforcement learning. In *Conference on Neural Information Processing Systems*, 2022.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. AWAC: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Mitsuhiko Nakamoto, Yuexiang Zhai, Anikait Singh, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. Cal-QL: Calibrated offline RL pre-training for efficient online fine-tuning. In *Workshop on Reincarnating Reinforcement Learning at International Conference on Learning Representations*, 2023.

Seohong Park, Qiyang Li, and Sergey Levine. Flow q-learning. *arXiv preprint arXiv:2502.02538*, 2025.

Panagiotis Petsagkourakis, Ilya Orson Sandoval, Eric Bradford, Dongda Zhang, and Ehecatl Antonio del Rio-Chanona. Reinforcement learning for batch bioprocess optimization. *Computers & Chemical Engineering*, 133:106649, 2020.

Nagabhushanamgari Rajasekhar, Thota Karunakaran Radhakrishnan, and Samsudeen Naina Mohamed. Reinforcement learning based temperature control of a fermentation bioreactor for ethanol production. *Biotechnology and Bioengineering*, 121(10):3114–3127, 2024.

Yongjae Shin, Jeonghye Kim, Whiyoung Jung, Sunghoon Hong, Deunsol Yoon, Youngsoo Jang, Geonhyeong Kim, Jongseong Chae, Youngchul Sung, Kanghoon Lee, et al. Online pre-training for offline-to-online reinforcement learning. *arXiv preprint arXiv:2507.08387*, 2025.

Yongjae Shin, Jongseong Chae, Jongeui Park, and Youngchul Sung. Flow matching with injected noise for offline-to-online reinforcement learning. *arXiv preprint arXiv:2602.18117*, 2026.

Laura Smith, J. Chase Kew, Xue Bin Peng, Sehoon Ha, Jie Tan, and Sergey Levine. Legged robots that keep on learning: Fine-tuning locomotion policies in the real world. In *International Conference on Robotics and Automation*, 2022.

Yuda Song, Yifei Zhou, Ayush Sekhari, Drew Bagnell, Akshay Krishnamurthy, and Wen Sun. Hybrid RL: Using both offline and online data can make RL efficient. In *International Conference on Learning Representations*, 2023.

Denis Tarasov, Vladislav Kurenkov, Alexander Nikulin, and Sergey Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. *Advances in Neural Information Processing Systems*, 36:11592–11620, 2023.

Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High confidence policy improvement. In *International Conference on Machine Learning*, pp. 2380–2388. PMLR, 2015.

Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, Sergey Levine, and Karol Hausman. Jump-start reinforcement learning. In *International Conference on Machine Learning*, 2023.

Ruosong Wang, Dean P Foster, and Sham M Kakade. What are the statistical limits of offline rl with linear function approximation? *International Conference on Learning Representations*, 2021.

Maciej Wołczyk, Bartłomiej Cupiał, Mateusz Ostaszewski, Michał Bortkiewicz, Michał Zajac, Razvan Pascanu, Łukasz Kuciński, and Piotr Miłoś. Fine-tuning reinforcement learning models is secretly a forgetting mitigation problem. *arXiv preprint arXiv:2402.02868*, 2024.

Chenjun Xiao, Han Wang, Yangchen Pan, Adam White, and Martha White. The in-sample softmax for offline reinforcement learning. In *International Conference on Learning Representations*, 2023.

Haeun Yoo, Ha Eun Byun, Dongho Han, and Jay H Lee. Reinforcement learning for batch process control: Review and perspectives. *Annual Reviews in Control*, 52:108–119, 2021.

Haichao Zhang, Wei Xu, and Haonan Yu. Policy expansion for bridging offline-to-online reinforcement learning. In *International Conference on Learning Representations*, 2023.

Yinmin Zhang, Jie Liu, Chuming Li, Yazhe Niu, Yaodong Yang, Yu Liu, and Wanli Ouyang. A perspective of q-value estimation on offline-to-online reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Kai Zhao, Yi Ma, Jinyi Liu, HAO Jianye, Yan Zheng, and Zhaopeng Meng. Improving offline-to-online reinforcement learning with q-ensembles. In *International Conference on Machine Learning Workshop on New Frontiers in Learning, Control, and Dynamical Systems*, 2023.

Yi Zhao, Rinu Boney, Alexander Ilin, Juho Kannala, and Joni Pajarinen. Adaptive behavior cloning regularization for stable offline-to-online reinforcement learning. *arXiv preprint arXiv:2210.13846*, 2022.

Zhiyuan Zhou, Andy Peng, Qiyang Li, Sergey Levine, and Aviral Kumar. Efficient online reinforcement learning fine-tuning need not retain offline data. In *International Conference on Learning Representations*, 2025.

Appendix

TABLE OF CONTENTS

A Theoretical Results **14**

B Related Work **18**

C Experimental Details **19**

D The COE Algorithm **21**

E Additional results **23**
E.1 The utility of normalization 23
E.2 COE Hyperparameter Study 23
E.3 Learning curves 24
E.4 Other Analysis Figures 24

F Investigating Why Performance Degradation Happens **32**
F.1 Encountering New States and Actions During Fine-tuning is Problematic 32
F.2 Approaches that did not prevent degradation 33

A THEORETICAL RESULTS

Proofs for all theoretical results use the setting and definitions of Section 6.

Theorem 6.1 (Impossibility of universal zero degradation)**.** *Fix $K \geq 1$. No offline-to-online learning algorithm can, for every $M \in \mathcal{M}$ and every finite offline dataset that can arise in M, simultaneously satisfy:*

(i) ***Zero degradation:*** $\mathbb{P}_M\big(J_M(\pi^{(k)}) \geq J_M(\pi^{(0)})\ \forall k = 1, \ldots, K\big) = 1$*;*

(ii) ***Nontrivial improvement:*** *whenever there is a policy $\bar{\pi} \in \Pi_M$ where $J_M(\bar{\pi}) > J_M(\pi^{(0)})$, the algorithm satisfies $\mathbb{P}_M\big(J_M(\pi^{(k)}) > J_M(\pi^{(0)})$ for some $k = 1, \ldots, K\big) > 0$.*

Proof. Suppose an algorithm satisfies both properties. Consider two one-step MDPs $M_+, M_- \in \mathcal{M}$ with initial state s and terminal states s^0, s^1. Choose two distinct actions a_0, a_1. Let $R(s, a, s^0) = 0,\ R(s, a, s^1) = R_{\max}$ for every action a. Define

$$P_\pm(s^1 \mid s, a_0) = \frac{1}{2}, \qquad P_+(s^1 \mid s, a_1) = \frac{3}{4}, \qquad P_-(s^1 \mid s, a_1) = \frac{1}{4},$$

with the remaining probability assigned to s^0. If the action space contains additional actions, give each of them the same transition law as a_0. Every action terminates the episode. Thus,

$$\mathbb{E}_\pm[R \mid a_0] = \frac{R_{\max}}{2}, \qquad \mathbb{E}_+[R \mid a_1] = \frac{3R_{\max}}{4}, \qquad \mathbb{E}_-[R \mid a_1] = \frac{R_{\max}}{4}.$$

Fix any offline behavior policy that selects both a_0 and a_1 with positive probability, and fix a finite realized dataset $\mathcal{D}_{\text{off}}$ containing samples of both actions. Because both terminal outcomes have strictly positive probability under both MDPs for every action, the same realized dataset has positive probability under both M_+ and M_-. As in the theorem statement, condition throughout on this realized dataset and initial policy $\pi^{(0)}$. Let $\mathbb{P}_\pm$ denote the resulting conditional laws under $M_\pm$, let $J_\pm := J_{M_\pm}$, and define $p_k := \pi^{(k)}(a_1 \mid s)$. In particular, p_0 is fixed under this conditioning. Since all actions other than a_1 have expected reward $R_{\max}/2$, for every policy π,

$$J_\pm(\pi) = \frac{R_{\max}}{2} \pm \frac{R_{\max}}{4}\pi(a_1 \mid s).$$

Define

$$A := \{\exists k \in \{1, \ldots, K\} : p_k < p_0\}, \qquad B := \{\exists k \in \{1, \ldots, K\} : p_k > p_0\}.$$

Since $J_+(\pi)$ is strictly increasing in $\pi(a_1 \mid s)$, the zero degradation property in M_+ implies $\mathbb{P}_+(A) = 0$. Similarly, since $J_-(\pi)$ is strictly decreasing in $\pi(a_1 \mid s)$, zero degradation in M_- implies $\mathbb{P}_-(B) = 0$.

For $n \geq 0$, let $\mathcal{F}_n$ contain the first n online transitions together with the learner's internal randomness revealed up to that point. Given the same observed history, the algorithm uses the same decision and update rules under M_+ and M_- (since these rules depend only on the offline dataset, observed interaction, and internal randomness). Also, conditional on any action, every possible terminal outcome has strictly positive probability under both MDPs. It follows inductively that, for every finite n, the restrictions of $\mathbb{P}_+$ and $\mathbb{P}_-$ to $\mathcal{F}_n$ are mutually absolutely continuous. So, for every $E \in \mathcal{F}_n$, $\mathbb{P}_+(E) = 0 \iff \mathbb{P}_-(E) = 0$.

Let N_K be the number of online transitions observed by the time the first K updates have been produced. By assumption, $N_K < \infty$ almost surely. The occurrence of learner's first K updates by time n is determined by $\mathcal{F}_n$, so N_K is a stopping time with respect to $(\mathcal{F}_n)_{n\geq 0}$. If E is any event determined by the first K updates, then

$$E = \bigcup_{n=0}^{\infty} \left(E \cap \{N_K = n\}\right),$$

where each $E \cap \{N_K = n\} \in \mathcal{F}_n$. So, if $\mathbb{P}_+(E) = 0$, then for every n, $\mathbb{P}_+(E \cap \{N_K = n\}) = 0$, and we get $\mathbb{P}_-(E \cap \{N_K = n\}) = 0$. Taking the countable union gives $\mathbb{P}_-(E) = 0$. Reversing the roles of M_+ and M_- gives the converse. So, for every event determined by the first K updates, $\mathbb{P}_+(E) = 0 \iff \mathbb{P}_-(E) = 0$.

Applying this to A and B gives $\mathbb{P}_-(A) = 0$, $\mathbb{P}_+(B) = 0$. With $\mathbb{P}_+(A) = 0$ and $\mathbb{P}_-(B) = 0$, both events have probability zero under either MDPs. So, $\mathbb{P}_\pm(A \cup B) = 0$. On $(A \cup B)^c$, neither $p_k < p_0$ nor $p_k > p_0$ occur for any $k = 1, \ldots, K$. Therefore,

$$\mathbb{P}_\pm(p_k = p_0 \text{ for all } k = 1, \ldots, K) = 1.$$

It follows that

$$\mathbb{P}_\pm\Big(J_\pm(\pi^{(k)}) > J_\pm(\pi^{(0)}) \text{ for some } k = 1, \ldots, K\Big) = 0.$$

Let $\bar{\pi}_1$ and $\bar{\pi}_0$ be the deterministic policies that choose a_1 and a_0, respectively, at s. Then,

$$J_+(\bar{\pi}_1) - J_+(\pi^{(0)}) = \frac{R_{\max}(1 - p_0)}{4}, \qquad J_-(\bar{\pi}_0) - J_-(\pi^{(0)}) = \frac{R_{\max} p_0}{4}.$$

For every $p_0 \in [0, 1]$, at least one of these quantities is strictly positive. So at least one of M_+ or M_- admits a policy in Π_M that strictly improves on $\pi^{(0)}$, while the algorithm has zero probability of producing a strict improvement within its first K updates in either MDPs. So, we have contradicted the nontrivial-improvement property.

□

Theorem 6.2 (Backwards expansion minimizes worst-case degradation)**.** *For every* $1 \le d \le H$*,*

$$\operatorname*{arg\,min}_{\mathcal{I} \subseteq \{0, \ldots, H-1\}, |\mathcal{I}| = d} \mathcal{R}(\mathcal{I}) = \{\mathcal{I}_d^\star\}, \qquad \textit{where } \mathcal{I}_d^\star = \{H - d, \ldots, H - 1\}.$$

Proof. Fix a nonempty $\mathcal{I} \subseteq \{0, \ldots, H-1\}$ and let $m = \min \mathcal{I}$. Fix any $M \in \mathcal{M}$ and $\pi^c, \pi^e \in \Pi_M$. Since $\pi_\mathcal{I}$ and π^c agree at every time $t < m$, they induce the same trajectory distribution up to time m. Hence $\mathbb{E}_M^{\pi_\mathcal{I}}[r_{t+1}] = \mathbb{E}_M^{\pi^c}[r_{t+1}]$, $t < m$. Therefore,

$$\begin{aligned}
[J_M(\pi^c) - J_M(\pi_\mathcal{I})]_+ &\le |J_M(\pi_\mathcal{I}) - J_M(\pi^c)| \\
&= \left| \sum_{t=m}^{H-1} \gamma^t \left(\mathbb{E}_M^{\pi_\mathcal{I}}[r_{t+1}] - \mathbb{E}_M^{\pi^c}[r_{t+1}] \right) \right| \\
&\le \sum_{t=m}^{H-1} \gamma^t \left| \mathbb{E}_M^{\pi_\mathcal{I}}[r_{t+1}] - \mathbb{E}_M^{\pi^c}[r_{t+1}] \right| \\
&\le 2R_{\max} \sum_{t=m}^{H-1} \gamma^t.
\end{aligned}$$

Since this holds for every $M \in \mathcal{M}$ and $\pi^c, \pi^e \in \Pi_M$, $\mathcal{R}(\mathcal{I}) \le 2R_{\max} \sum_{t=m}^{H-1} \gamma^t$.

To show that the bound is tight, consider an H-step deterministic MDP in $\mathcal{M}$ whose initial-state distribution is concentrated on s_0. Before time m, let the process follow a deterministic chain $s_0, \ldots, s_m$, independent of the actions taken, with zero reward. At s_m, let actions a^+ and a^- produce rewards $R_{\max}$ and $-R_{\max}$, respectively, and enter two branches that continue to produce $R_{\max}$ and $-R_{\max}$ at every remaining time step. Let $\pi^c(a^+ \mid s_m) = 1$, $\pi^e(a^- \mid s_m) = 1$, define both policies arbitrarily elsewhere, and let actions within each branch have no effect on subsequent rewards. Since $m \in \mathcal{I}$, $\pi_\mathcal{I}$ selects a^- at time m, whereas π^c selects a^+. Thus,

$$J_M(\pi^c) = R_{\max} \sum_{t=m}^{H-1} \gamma^t, \qquad J_M(\pi_\mathcal{I}) = -R_{\max} \sum_{t=m}^{H-1} \gamma^t.$$

Therefore,

$$[J_M(\pi^c) - J_M(\pi_\mathcal{I})]_+ = 2R_{\max} \sum_{t=m}^{H-1} \gamma^t.$$

Since this choice of M, π^c, π^e is included in the supremum defining $\mathcal{R}(\mathcal{I})$, we have that

$$\mathcal{R}(\mathcal{I}) \ge 2R_{\max} \sum_{t=m}^{H-1} \gamma^t.$$

Combining the upper and lower bounds gives $\mathcal{R}(\mathcal{I}) = 2R_{\max} \sum_{t=\min \mathcal{I}}^{H-1} \gamma^t$.

Now restrict to $|\mathcal{I}| = d$. Since $\mathcal{I} \subseteq \{m, \dots, H-1\}$, $d = |\mathcal{I}| \leq H - m$, and so $m \leq H - d$. And, because $\gamma > 0$,

$$\sum_{t=m+1}^{H-1} \gamma^t = \sum_{t=m}^{H-1} \gamma^t - \gamma^m < \sum_{t=m}^{H-1} \gamma^t.$$

So, $\mathcal{R}(\mathcal{I})$ is strictly decreasing in $m = \min \mathcal{I}$. Minimizing $\mathcal{R}(\mathcal{I})$ over all $|\mathcal{I}| = d$ is therefore equivalent to maximizing $\min \mathcal{I}$, whose largest possible value is $H - d$. If $\min \mathcal{I} = H - d$, then $\mathcal{I} \subseteq \{H-d, \dots, H-1\}$. Since both sets have cardinality d, we have that $\mathcal{I} = \{H-d, \dots, H-1\} = \mathcal{I}_d^\star$. So, $\mathcal{I}_d^\star$ is the unique minimizer, and

$$\underset{\mathcal{I} \subseteq \{0,\dots,H-1\}, |\mathcal{I}|=d}{\arg\min} \mathcal{R}(\mathcal{I}) = \{\mathcal{I}_d^\star\}.$$

□

Proposition 1 (Finite-time rollback under persistent degradation)**.** *Let c be a rollback threshold fixed by the pre-expansion history. Suppose episodic returns lie in an interval of width $B > 0$. If, while the expansion remains deployed, the conditional expected return of each post-expansion episode is at least $\varepsilon > 0$ below c. Then a rollback test using the mean of m post-expansion returns satisfies, conditional on the pre-expansion history,*

$$\mathbb{P}(\textit{no rollback within } m \textit{ episodes}) \leq \exp\left(-\frac{2m\varepsilon^2}{B^2}\right).$$

Proof. Condition on the pre-expansion history and denote its sigma-field by $\mathcal{F}_0$. Conditional on $\mathcal{F}_0$, the rollback threshold c is fixed. For $i = 1, \dots, m$, let Z_i denote the return of the ith post-expansion episode, and let $\mathcal{F}_i$ be the sigma-field generated by $\mathcal{F}_0$ together with all interaction observed through post-expansion episode i. Define $\mu_i := \mathbb{E}[Z_i \mid \mathcal{F}_{i-1}]$.

Let τ be the index of the first post expansion episode after which rollback is triggered, with $\tau = \infty$ if it is not triggered. Since the rollback decision following episode i is determined by $\mathcal{F}_i$, τ is a stopping time with respect to $(\mathcal{F}_i)_{i \geq 0}$. So, $\{\tau \geq i\} = \{\tau > i - 1\} \in \mathcal{F}_{i-1}$. By assumption, while the expansion stays deployed, conditional expected return of each episode is at most $c - \varepsilon$. So, $\mu_i \leq c - \varepsilon$ on $\{\tau \geq i\}$, $i = 1, \dots, m$.

Define $X_i := \mathbb{I}_{\{\tau \geq i\}}(Z_i - \mu_i)$, $S_j := \sum_{i=1}^{j} X_i$, $S_0 = 0$. Each X_i is $\mathcal{F}_i$-measurable and integrable. And, because $\mathbb{I}_{\{\tau \geq i\}}$ is $\mathcal{F}_{i-1}$-measurable,

$$\mathbb{E}[X_i \mid \mathcal{F}_{i-1}] = \mathbb{I}_{\{\tau \geq i\}} \mathbb{E}[Z_i - \mu_i \mid \mathcal{F}_{i-1}] = \mathbb{I}_{\{\tau \geq i\}} (\mathbb{E}[Z_i \mid \mathcal{F}_{i-1}] - \mu_i) = 0.$$

So, $(S_j, \mathcal{F}_j)_{j=0}^m$ is a martingale. Now, by assumption, episodic returns lie almost surely in some interval $[L, U]$ with $U - L = B$. So, $\mu_i = \mathbb{E}[Z_i \mid \mathcal{F}_{i-1}] \in [L, U]$ a.s. Conditional on $\mathcal{F}_{i-1}$, the event $\{\tau \geq i\}$ is known. If $\tau < i$, then $X_i = 0$. If $\tau \geq i$, then $X_i = Z_i - \mu_i \in [L - \mu_i, U - \mu_i]$.

Since $\mu_i \in [L, U]$, the interval $[L - \mu_i, U - \mu_i]$ contains zero and has width B. So, conditional on $\mathcal{F}_{i-1}$, X_i lies almost surely in an interval of width at most B. Since $\mathbb{E}[X_i \mid \mathcal{F}_{i-1}] = 0$, the conditional form of Hoeffding's lemma gives, for every $\lambda > 0$,

$$\mathbb{E}\left[e^{\lambda X_i} \mid \mathcal{F}_{i-1}\right] \leq \exp\left(\frac{\lambda^2 B^2}{8}\right).$$

Using $S_i = S_{i-1} + X_i$ and iterating this inequality gives us

$$\mathbb{E}\left[e^{\lambda S_m} \mid \mathcal{F}_0\right] \leq \exp\left(\frac{m\lambda^2 B^2}{8}\right).$$

Now consider the event $\{\tau > m\}$, corresponding to no rollback within the first m post-expansion episodes. Since the rollback test after episode m triggers whenever $\frac{1}{m} \sum_{i=1}^m Z_i < c$, the event $\{\tau > m\}$ implies $\sum_{i=1}^m Z_i \geq mc$. And, on $\{\tau > m\}$ we have $\tau \geq i$ for every $i = 1, \dots, m$. So,

$\mu_i \le c - \varepsilon,\ i = 1, \dots, m$, and so $\sum_{i=1}^{m} \mu_i \le m(c - \varepsilon)$. Also, $\mathbb{I}_{\{\tau \ge i\}} = 1$ for every $i \le m$ on this event, so $X_i = Z_i - \mu_i$. Consequently,

$$S_m = \sum_{i=1}^{m} (Z_i - \mu_i) \ge mc - m(c - \varepsilon) = m\varepsilon.$$

Thus $\{\tau > m\} \subseteq \{S_m \ge m\varepsilon\}$. For any $\lambda > 0$, conditional Markov's inequality gives us

$$\begin{aligned}
\mathbb{P}(\tau > m \mid \mathcal{F}_0) &\le \mathbb{P}(S_m \ge m\varepsilon \mid \mathcal{F}_0) \\
&= \mathbb{P}\big(e^{\lambda S_m} \ge e^{m\lambda\varepsilon} \mid \mathcal{F}_0\big) \\
&\le e^{-m\lambda\varepsilon} \mathbb{E}\big[e^{\lambda S_m} \mid \mathcal{F}_0\big] \\
&\le \exp\left(-m\lambda\varepsilon + \frac{m\lambda^2 B^2}{8}\right).
\end{aligned}$$

And the exponent is minimized over $\lambda > 0$ at $\lambda^\star = 4\varepsilon / B^2$. Substituting gives

$$\mathbb{P}(\tau > m \mid \mathcal{F}_0) \le \exp\left(-\frac{2m\varepsilon^2}{B^2}\right).$$

Since $\{\tau > m\}$ is exactly the event of no rollback within the first m post-expansion episodes, this proves the claim.

□

B Related Work

Proposed remedies fall into three broad strategies. **Conservative learning** limits how far values or policies can depart from what the data supports: in-sample algorithms never bootstrap from unseen actions (Kostrikov et al., 2022; Xiao et al., 2023), PARS penalizes values of infeasible actions to suppress extrapolation (Kim et al., 2025), Cal-QL calibrates the offline values of CQL (Kumar et al., 2020) (Nakamoto et al., 2023), SO2 reduces Q-value estimation error during fine-tuning with perturbed value targets and more frequent critic updates (Zhang et al., 2024), and policy-constraint methods keep the fine-tuned policy close to its predecessor (Li et al., 2023; Luo et al., 2023). These approaches can reduce degradation, but the same conservatism limits how quickly the agent can improve online. **Buffer management** controls what the agent trains on, by mixing or rebalancing offline and online data (Lee et al., 2021; Ball et al., 2023; Song et al., 2023), or by discarding the offline data after a short warm-up of rollouts from the pretrained policy (Zhou et al., 2025). **Policy composition** maintains separate policies for stability and for improvement and switches between them. PEX keeps the offline policy frozen and selects between it and a new policy using action values (Zhang et al., 2023); JSRL lets a fixed guide policy act for the first steps of each episode and hands control to an exploration policy, shrinking the guide's share on a curriculum (Uchendu et al., 2023). Our approach combines conservative learning with policy composition. Unlike JSRL, which targets exploration, we target improvement without degradation: the conservative policy continues to learn in deployment, the horizon controlled by the exploratory policy expands or contracts based on online performance estimates, and low-value actions are filtered.

There has been some work also trying to measure the balance between two objectives related to policy degradation and performance gain. Li et al. (2025) investigated a stability-plasticity trade-off in offline-online RL. They measured stability as the difference in performance during online fine-tuning to either the offline learned policy or the dataset policy (whichever was best). The plasticity is measured as the difference between the best and worst policies found during online fine-tuning. Their stability measure is similar to degradation, and they often use the two terms interchangeably. They found many algorithms suffered from degradation, especially when adjusted to have much higher plasticity.

Relation to safe policy improvement. Our goal of improving on the offline policy without first degrading it is related to safe policy improvement, which seeks policies that, with high probability, perform no worse than a baseline. High-confidence policy improvement (Thomas et al., 2015) accepts a candidate policy only when a high-confidence off-policy estimate shows it outperforms the baseline. SPIBB (Laroche et al., 2019) falls back to the baseline policy in state-action pairs with insufficient data. These methods operate in the batch setting and produce a single improved policy with guarantees, whereas we consider online fine-tuning, where the agent continues to interact and the risk lies in performance during learning. COE shares SPIBB's intuition of deferring to a trusted policy where the learned policy is unreliable, but applies it over time.

Recent high-performance offline-to-online methods. Several recent methods report strong offline-to-online results. Q-chunking (Li et al., 2026) learns a critic over sequences of actions to speed up value propagation and exploration. Its performance gains are largest on long-horizon, sparse-reward manipulation tasks, which differ from the D4RL tasks we study. Adaptive Q-chunking (Gireesh et al., 2026) extends Q-chunking by adapting the chunk length. OPT (Shin et al., 2025) trains a new value function in a short online phase before fine-tuning. On the D4RL tasks OPT's final performance is comparable to PARS (Kim et al., 2025), which we include instead as a representative baseline. FQL (Park et al., 2025) and FINO (Shin et al., 2026) use expressive flow-based policies, and FINO additionally injects noise into flow matching to broaden exploration online; the choice of policy class is orthogonal to the question we study, and either could serve as a component within COE. In addition, neither has been compared against PARS, so it is unknown whether either improves on it.

C EXPERIMENTAL DETAILS

We describe the full experimental details here for easy reference, even though this results in repeating some of what is laid out in the main text.

For offline training, each algorithm optimizes for 10^6 gradient steps for each task. All networks are two layer MLPs of width 256 units. Optimization is done using Adam at 3×10^{-4}, $\gamma = 0.99$, Polyak rate 0.005 and batches of 256.

InAC (Xiao et al., 2023) updates the policy using only actions in the dataset. PARS (Kim et al., 2025) uses a strategy that penalizes infeasible actions and builds on the TD3+BC (Fujimoto & Gu, 2021) update instead of the SAC update. It uses similar objects and updates as SAC and InAC—learns action-values off-policy, uses a minimum over two action-values for the bootstrap target and uses reparameterized updates—but adds several additional tricks to get even more effective offline learning, outperforming previous related methods like ReBRAC (Tarasov et al., 2023). IQL uses an expectile loss to avoid bootstrapping on out-of-sample actions, providing a simple approach that also naturally extends to the online setting. CQL (Kumar et al., 2020) adds a component to the critic objective that reduces values for actions not in the dataset.

Online fine-tuning is done for 5×10^5 environment steps. Online fine-tuning begins from the offline checkpoint as follows: SAC from InAC, WSRL from Cal-QL, and COE from the InAC trained with behavior cloning. The remaining baselines finetune the policies trained by their own algorithms. RLPD trains online from scratch on offline data and loads no offline policy.

We tried to improve all the baselines and using simple tricks and reduce the confounding factors in empirical evaluation. All baselines use layer normalization, and observations normalized using dataset statistics. We use reward normalizations where we found them to be empirically beneficial as follows: SAC, InAC, Cal-QL and COE normalize rewards by dividing with the standard deviation of rewards within the dataset. IQL uses min-max reward normalization, and so does PEX since it finetunes the offline IQL checkpoint. PARS uses their own published reward scaling multiplier, and RLPD uses no reward normalization since it trains from scratch.

We define a 90-10 buffer for online fine-tuning. In this strategy, a fairly large buffer size is defined (2M in our work), and is filled in a first-in first-out manner. When sampling from this buffer, 50% of the batch is sampled from the most recent 10% of the buffer, and the remaining 50% of the batch is sampled from the earlier 90% of the buffer. We use this 90-10 buffer for InAC, COE, SAC, IQL, PEX, Cal-QL and PARS. RLPD and WSRL use their respective published buffer strategies. Finally, all algorithms use a replay ratio of 5 updates per step, and 2 critics.

Hyperparameters are lightly tuned for all methods, with final choices summarized in the Tables 3 and 2. We tuned hyperparameters on a small subset of environments: hc-mr, hopper-mr, hopper-exp, door-exp and pen-exp for 3 seeds each and choose the value with highest offline mean return. Afterwards, these tuning runs are discarded to remove bias due to hyperparameter evaluation. These hyperparameters are used across all environments for each algorithm, and the online phase does not change these hyperparameters.

Table 2: Common hyperparameters across all algorithms.

Hyperparameter	Value
Actor and critic networks	2-layer MLP, 256 units, LayerNorm
Learning rate	Adam with 10^{-4} for Cal-QL and CQL actors, 3×10^{-4} for everything else
Critic ensemble size E	2
Batch size	256
Discount γ	0.99
Target-network Polyak rate	0.005
Observation normalization	$(s - \mu_{\mathcal{D}})/\sigma_{\mathcal{D}}$ from the offline dataset
Offline pretraining	10^6 gradient updates
Online training steps	5×10^5 environment steps
Replay ratio	5

Table 3: Algorithm-specific hyperparameters of all methods. Highlighted values were tuned by us. All other values are derived from the original paper or its released code.

Alg.	Hyperparameter	Value
RLPD	offline pretraining	none
	replay	uniformly sample 50% offline, 50% online
	warm-up steps	5000
	target entropy	$-\lvert\mathcal{A}\rvert/2$
	reward scaling	none
WSRL	offline checkpoint	Cal-QL, $\alpha = 0.5$ (actor, critics, $\log\alpha$)
	warm-up steps with the frozen actor	5000
	target entropy	$-\lvert\mathcal{A}\rvert$
	reward scaling	divided by the dataset reward std
SAC	offline checkpoint	InAC, $\tau = 0.1$ (actor, critics and target)
	entropy coefficient	$\alpha_0 = 1$, target entropy $-\lvert\mathcal{A}\rvert$
	reward scaling	divided by the dataset reward std
InAC	temperature τ	$\{1.0, 0.5, 0.33, 0.1^*, 0.05, 0.01\}$
	BC anchor β_{bc}	0
	advantage-weight clip	$[10^{-8}, 100]$
	reward scaling	divided by the dataset reward std
IQL	expectile τ_e, inverse temperature β	$\{0.5, 0.7^*, 0.9\} \times \{3.0^*, 0.5\}$
	reward scaling	return-range: $r \cdot H_{\max}/(R_{\max} - R_{\min})$
PEX	offline checkpoint	IQL above: actor frozen; second actor randomly initialized, (actor, critic and value net)
	online τ_e, β	0.7, 0.5
Cal-QL	conservatism weight α	$\{0.5, 1, 5^*, 10\}$
	sampled actions k, temperature	10, 1.0
	policy learning rate	10^{-4}
	reward scaling	{divided by the dataset reward std*, return-range}
PARS	BC weight β	$\{0, 0.01, 0.03, 0.1^*, 0.3, 1.0\}$
	reward scale c_{reward}	10
	infeasible-action penalty α, range w	0.01, 100
	reward scaling	{divided by the dataset reward std, return-range, none*}
CQL	conservatism weight α	$\{0.5, 1, 2, 3, 5^*, 10\}$
	sampled actions k, temperature	10, 1.0
	policy learning rate	10^{-4}
	reward scaling	none
COE	checkpoint	InAC-BC: $\{\tau, \beta_{\text{bc}}\} : \{0.1^*, 0.01\} \times \{0.03, 0.1^*, 0.3\}$
	conservative policy π^c	InAC update, same parameters as checkpoint
	explorer policy π^e	SAC update, critic copied from the checkpoint, $\alpha_0 = 1$
	initial switching point h	H (the episode time limit)
	horizon release steps $n_{\min}$	250000
	FQE critics	$E = 2$ critic ensemble, lr 3×10^{-4}, Polyak 0.005
	FQE pretraining	blocks of 10^4 updates until $\hat{v}$ changes $< 1\%$, cap 5×10^5
	decline test: windows N / N_{rec}	30 / 5 episodes
JSRL	checkpoint	InAC-BC, same as COE
	conservative policy π^c	frozen offline policy
	explorer policy π^e	InAC update, $\tau = 0.1$, anchor $\beta_e = 0.01$
	curriculum test	{mean of the last 5 training returns within 5% of the running best*, means of last 5 training returns against 30 episode offline policy mean returns}
	curriculum step per passing episode	$\max(1, 2H^2/N)$, starting from $h = H$

D THE COE ALGORITHM

The pseudocode for COE is in Algorithm 1. We explain some additional details here.

Algorithm 1 Cautious Online Expansion (COE)

Input: Offline data $\mathcal{D}_{\text{off}}$, max horizon H, minimum steps n_{min} before caution horizon reaches 0 (default to half of fine-tuning phase), InAC algorithm with specified hyperparameters, SAC algorithm with specified hyperparameters

Learn offline policy with InAC, and initialize both conservative InAC policy π^c and critic and SAC policy π^e and critic with these offline learned estimates
Set hyperparameters $N = 30$ for running mean estimate of returns, $N_{\text{rec}} = 5$ for estimate of recent performance
Initialize replay buffer $\mathcal{B} \leftarrow \mathcal{D}_{\text{off}}$, empty return buffer $\mathcal{J} = \emptyset$, start state buffer $\mathcal{D}_0$ = start states in D_{off} and set $h \leftarrow H$
Estimate $\hat{Q}^{\pi^c}$ on $\mathcal{D}_{\text{off}}$ and initialize $\hat{Q}^{\pi^h}$ also to this estimate
Get start state s_0 and set $t \leftarrow 0$
while Agent interacting with the environment **do**
  Act with current composite policy:
  $a_t \sim \begin{cases} \pi^c(\cdot \mid s_t) \text{ with mode-screening}, & t < h, \\ \pi^e(\cdot \mid s_t), & t \geq h \end{cases}$
  Observe $r_{t+1}, s_{t+1}, \gamma_{t+1}, t$ and append $(s_t, a_t, r_{t+1}, s_{t+1}, t)$ to $\mathcal{B}$
  Update π^c with InAC, π^e with SAC, and both FQE critics
  if episode ends after T steps **then**
    Set $Z \leftarrow \sum_{t=0}^{T-1} \gamma^t r_{t+1}$, append Z to $\mathcal{J}$
    $\delta \leftarrow TH/n_{\text{min}}$
    Evaluate policies with FQE, using Equation 4:
    $\widehat{J}^h \leftarrow \hat{v}(\pi^h, \hat{Q}^{\pi^h}, \mathcal{D}_0), \quad \widehat{J}^c \leftarrow \hat{v}(\pi^c, \hat{Q}^{\pi^c}, \mathcal{D}_0)$
    If $|\mathcal{J}| \geq N + N_{\text{rec}}$, compute mean $\bar{Z}_{\text{rec}}$ of last N_{rec} returns, mean $\bar{Z}$ and std σ of preceding N returns
    Update boundary:
    if $|\mathcal{J}| \geq N + N_{\text{rec}}$ and $\bar{Z}_{\text{rec}} < \bar{Z} - \sigma/\sqrt{N}$ **then**
      $h \leftarrow \min\{H, h + \delta\}$ {shift control toward conservative policy}
    else if $\widehat{J}^h \geq \widehat{J}^c$ **then**
      $h \leftarrow \max\{0, h - \delta\}$ {shift control toward exploratory policy}
    Reset to new start state s_0, set $t = 0$, add s_0 to $\mathcal{D}_0$

The first is how the size of the step δ backwards (or forwards) is chosen. Simply setting this to 1 might be too slow for some environments. We use a rate of change, based on the fastest that a user might accept having the caution horizon reduced all the way to zero. Let n_{min} be the minimum number of steps in online fine-tuning that COE must be learning before it can have its caution horizon reduce to 0. For example, a user might know they are fine-tuning for 500k steps, and want to ensure that the caution horizon is not reduced to 0 any earlier than halfway through this learning, meaning they set $n_{\text{min}} = 250k$. Therefore, the rate of change per step is at most H/n_{min}, where if we decrease h by the fraction H/n_{min} every step, we would reach $h = H - n_{\text{min}}(H/n_{\text{min}}) = 0$ after n_{min} steps. Each episode can be a variable length T, so if h is reduced after an episode, we reduce it by TH/n_{min} to ensure we reduce it by a rate of H/n_{min} per step in the episode. Note that h is stored as a real-valued number, not an integer.

There are several other hyperparameters, for which we propose default settings. We use InAC with default hyperparameter settings: update-to-data ratio (UTD) = 5, entropy $\tau = 0.1$, behavior cloning parameter $\beta = 0.1$. SAC uses the same UTD and entropy. These hyperparmeters generalized more effectively across environments because of the observation and reward normalization and layernorm included in both InAC and SAC. See Table 2 for more details on hyperparameters.

Finally, the conservative policy π^c is updated online with the InAC actor loss $\mathcal{L}_{\text{InAC}}(\theta) = -\mathbb{E}_{(s,a)\sim\mathcal{B}}\big[w(s,a)\log\pi^c_\theta(a \mid s)\big]$ where w is the in-sample weight. In order to encourage the conservative policy to stay close to the dataset transitions, we compute a behavior cloning term toward the batch actions, $\mathcal{L}_{\text{BC}}(\theta) = \mathbb{E}_{(s,a)\sim\mathcal{B},\, \hat{a}\sim\pi^c_\theta(\cdot|s)}\big[\|\hat{a} - a\|^2\big]$, with $\hat{a}$ a reparameterized sample

of the policy. The two losses are combined after computing their gradients. Let $g_{\mathrm{InAC}} = \nabla_\theta \mathcal{L}_{\mathrm{InAC}}$ and $g_{\mathrm{BC}} = \nabla_\theta \mathcal{L}_{\mathrm{BC}}$ at the current parameters. The actor step uses

$$g = g_{\mathrm{InAC}} + \beta_{\mathrm{bc}} \frac{\|g_{\mathrm{InAC}}\|}{\|g_{\mathrm{BC}} + 10^{-8}\|} g_{\mathrm{BC}}$$

This allows the value of β_{bc} to be transferable across environments. We use $\beta_{\mathrm{bc}} = 0.1$ everywhere. $\beta_{\mathrm{bc}} = 0$ recovers plain InAC. Figure E.2 shows the sensitivity to β_{bc}.

E ADDITIONAL RESULTS

E.1 THE UTILITY OF NORMALIZATION

We report the utility of adding normalization to InAC. The base InAC is noisier and performs more poorly, both in terms of offline performance and online performance, as we see in Figure 7.

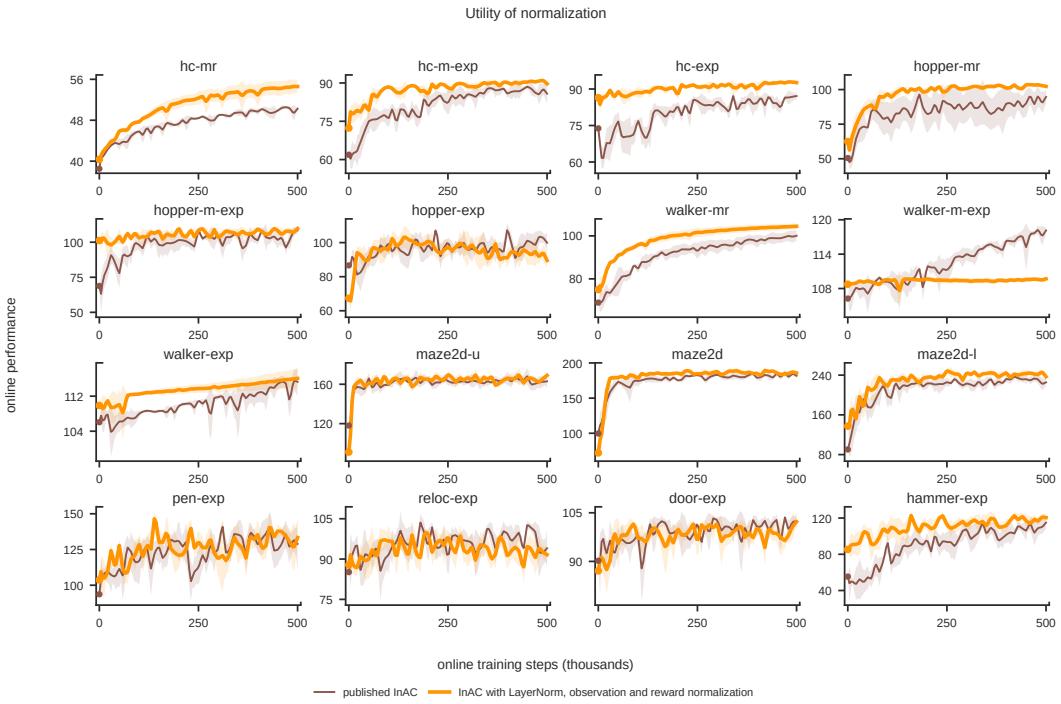


Figure 7: Comparison of InAC with and without observation and reward normalization, and without LayerNorm. The hyperparameters for both methods were tuned independently.

E.2 COE HYPERPARAMETER STUDY

We also conducted a hyperparameter study for COE. We had not swept these parameters, but instead retroactively investigated the sensitivity to the choices we made from first principles. We find that sensitivity curves are quite flat, indicating COE is not that sensitive to its hyperparameters. The most important choice was $n_{\min}$, where when it was too small, VaR-MPD became worse. This makes sense, as control may be handed off to the exploratory too quickly with a small $n_{\min}$.

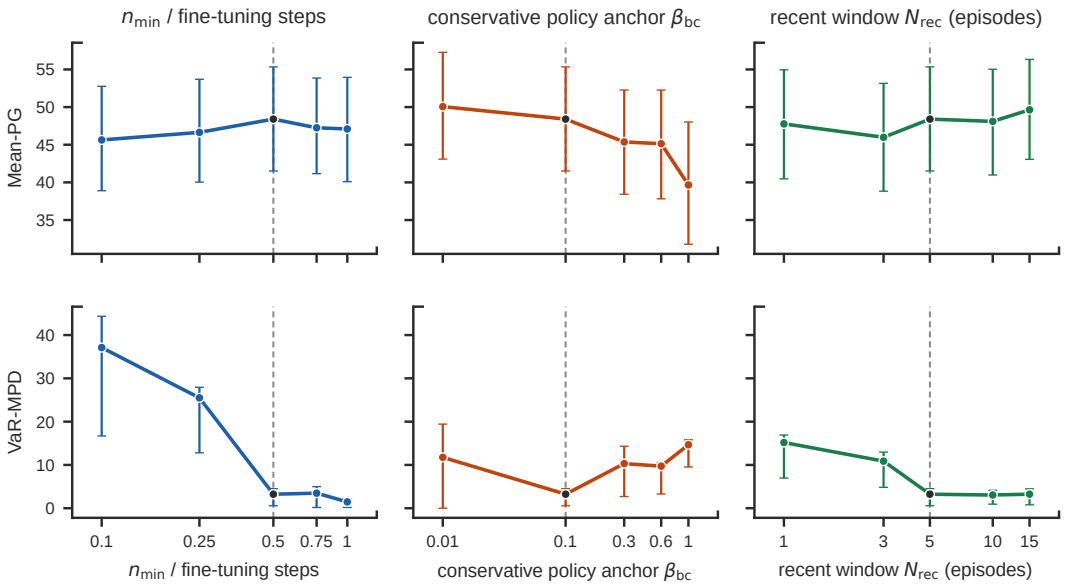


Figure 8: Hyperparameter sensitivity for COE for three hyperparameters, in terms of Mean-PG and VaR-MPD.

E.3 Learning curves

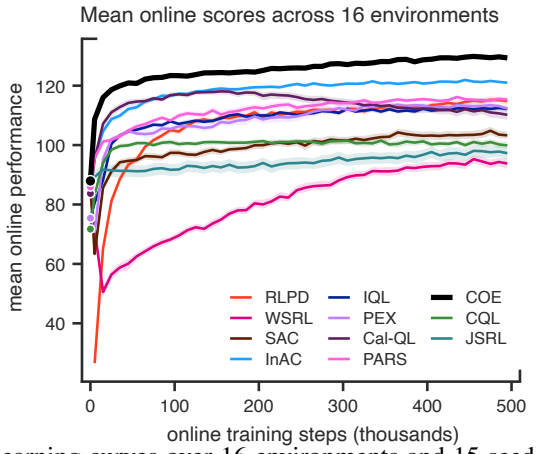


Figure 9: Mean online learning curves over 16 environments and 15 seeds. Each point is the normalized score averaged over a 10k step bin, after which the bins are averaged over seeds, and then over environments. These scores are computed using the stochastic policy taking actions during the training, rather than a separate evaluation environment. Bands indicate 95% stratified bootstrap confidence intervals.

For the comparison to other offline-to-online algorithms, we also examined the learning curves, averaged across all 16 settings. We see that COE learns notably faster than the other approaches in Figure 9.

E.4 Other Analysis Figures

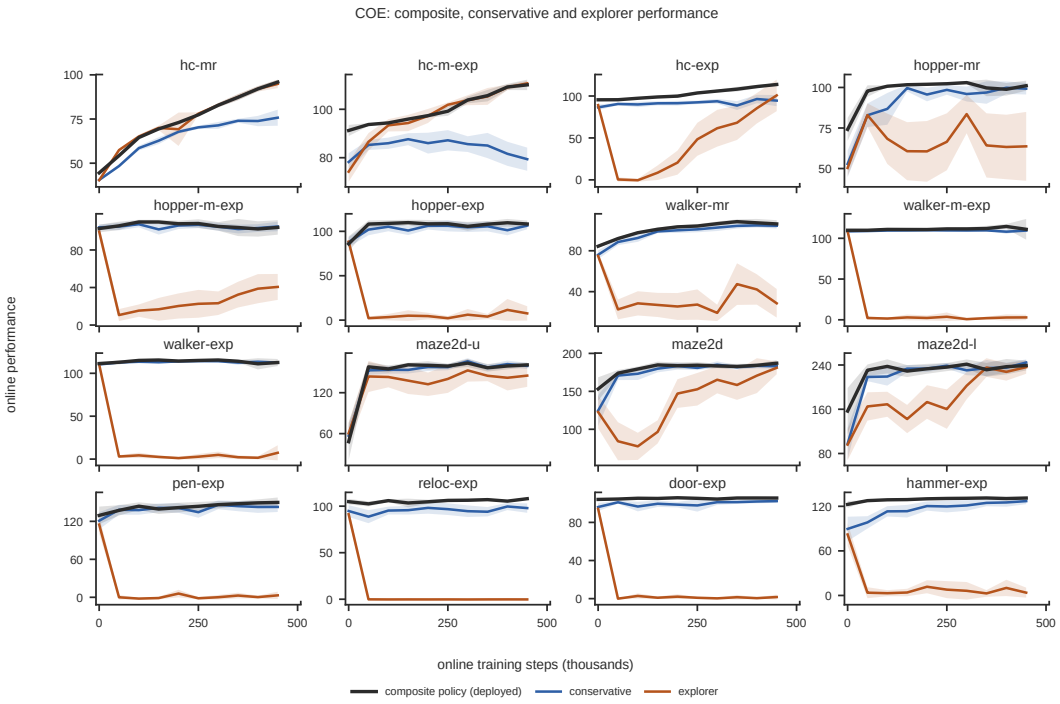


Figure 10: COE effectively contains three policies: conservative, explorer and the combined composite. We pause the training at repeated intervals and evaluate the performance of each policy on separate evaluation environments. We observe that the composite policy is able to pick up whichever underlying policy performs the best at any point in time.

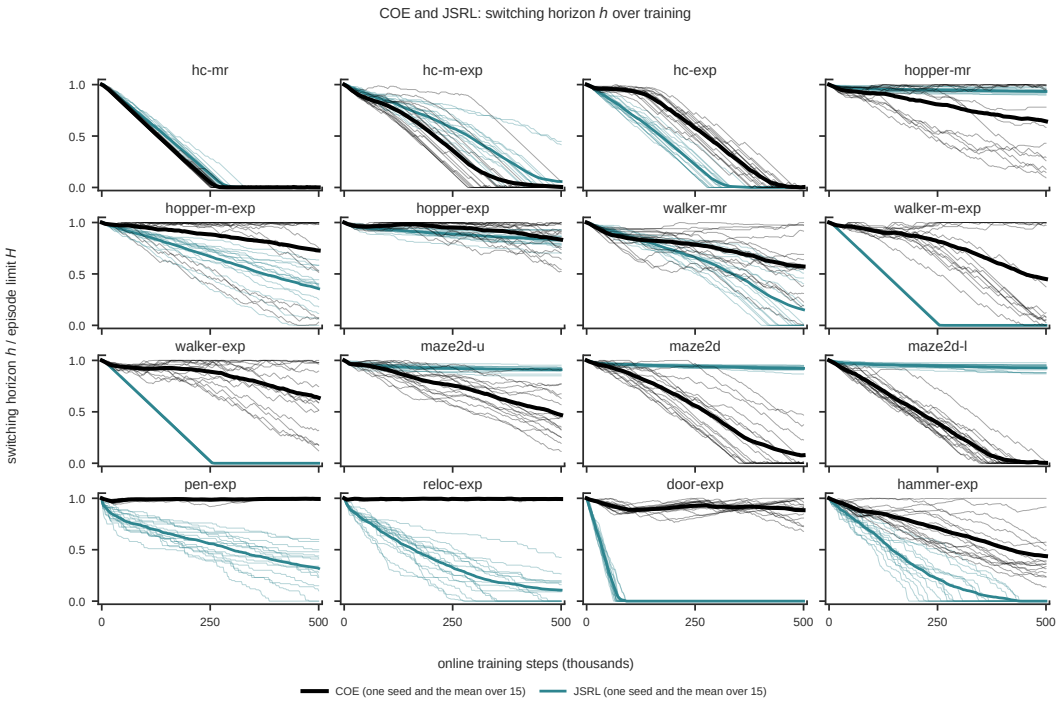


Figure 11: Comparison of switching horizon in COE and JSRL.

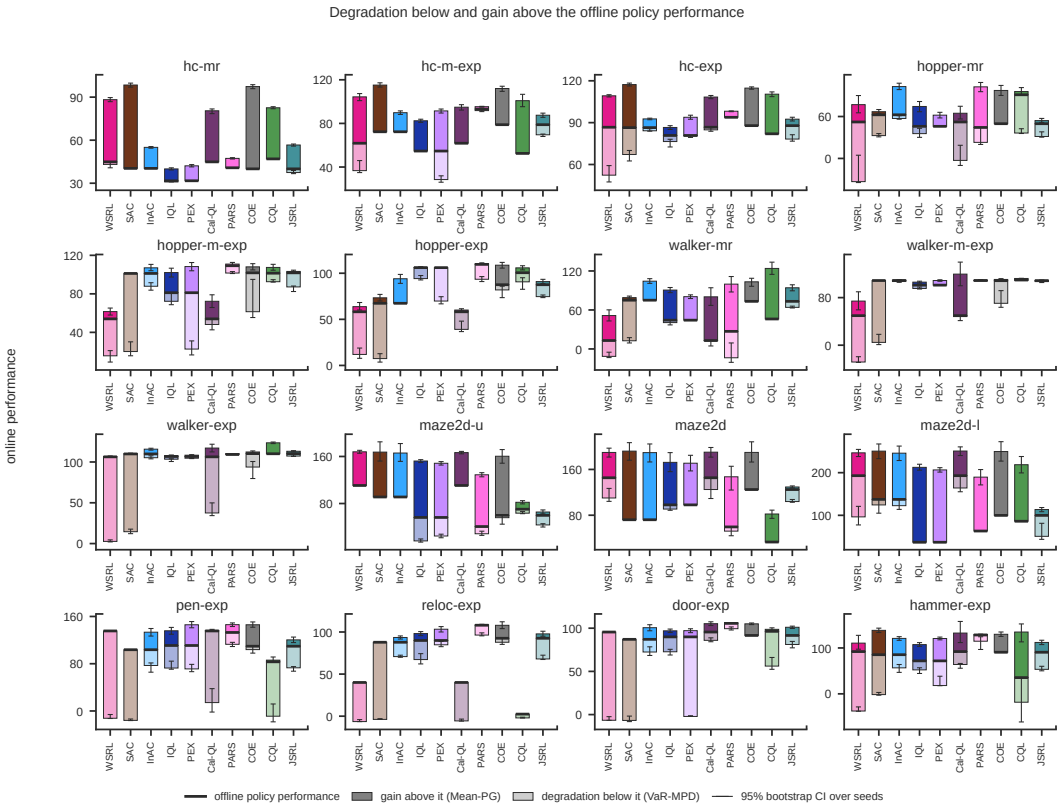


Figure 12: An overview of all algorithms used in our work. These are not barplots. The solid line in the middle of boxes represents the performance of offline policy, the bar on the top is the Mean-PG while the bar at the bottom is VaR-MPD.

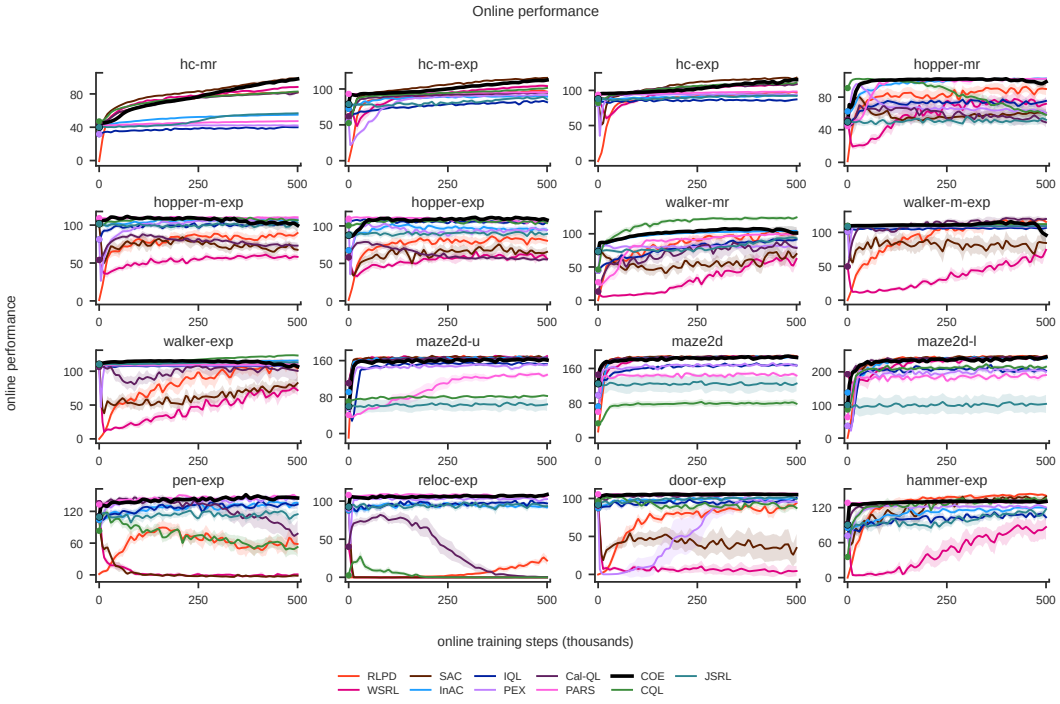


Figure 13: Online performance curves across all environments and algorithms.

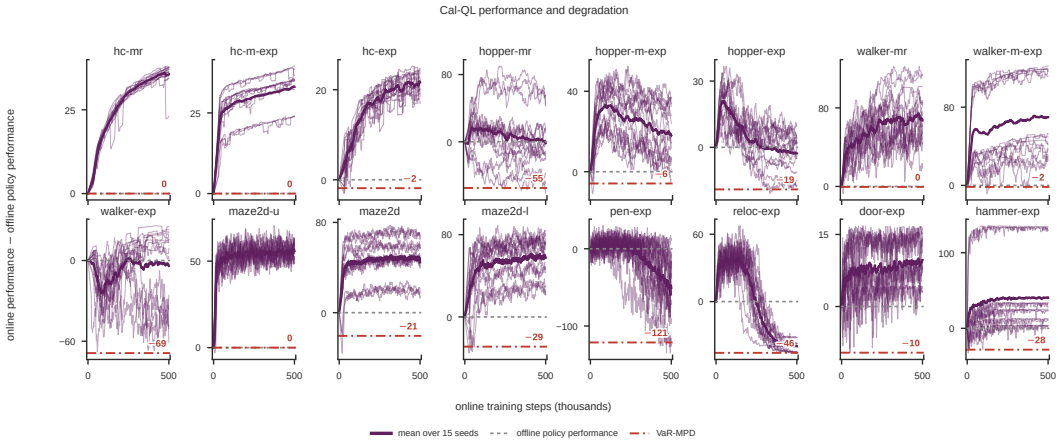


Figure 14: Cal-QL performance across all environments.

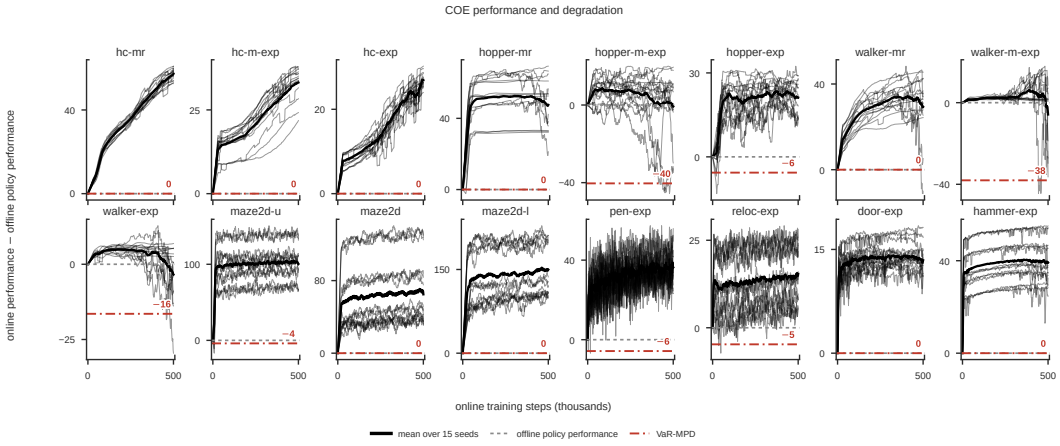


Figure 15: COE performance across all environments.

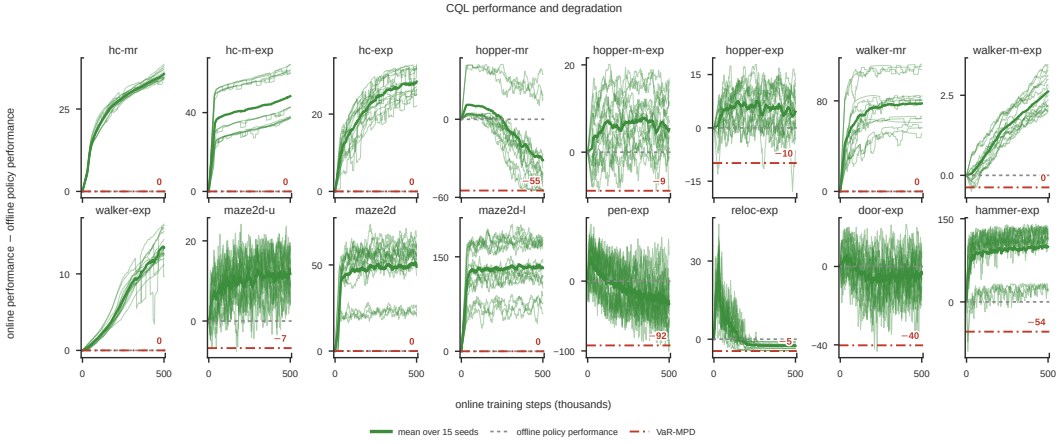


Figure 16: CQL performance across all environments.

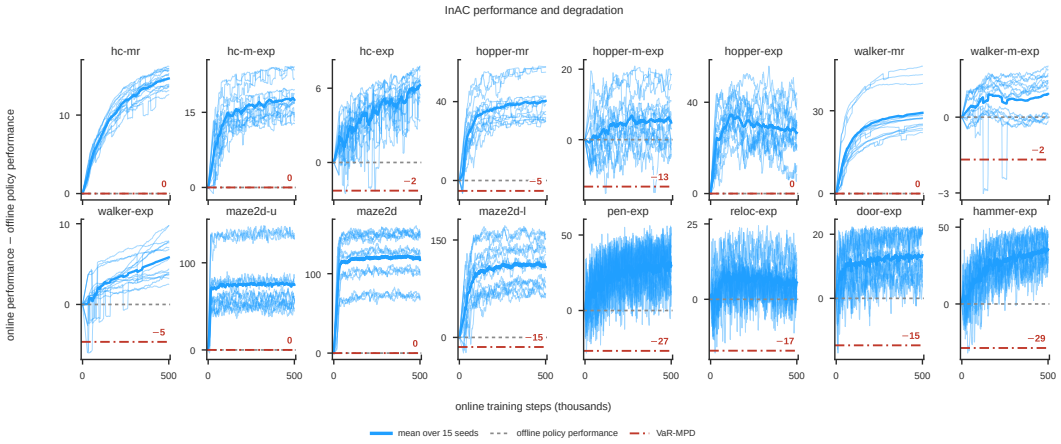


Figure 17: InAC performance across all environments.

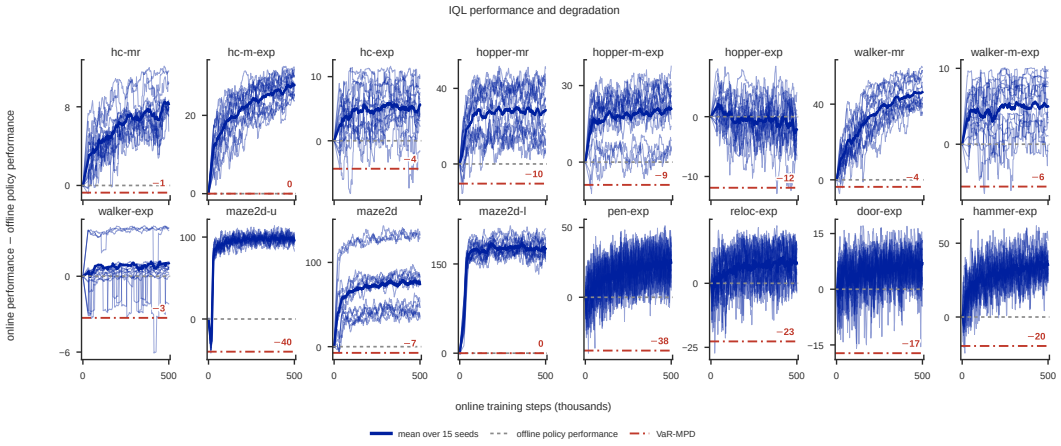


Figure 18: IQL performance across all environments.

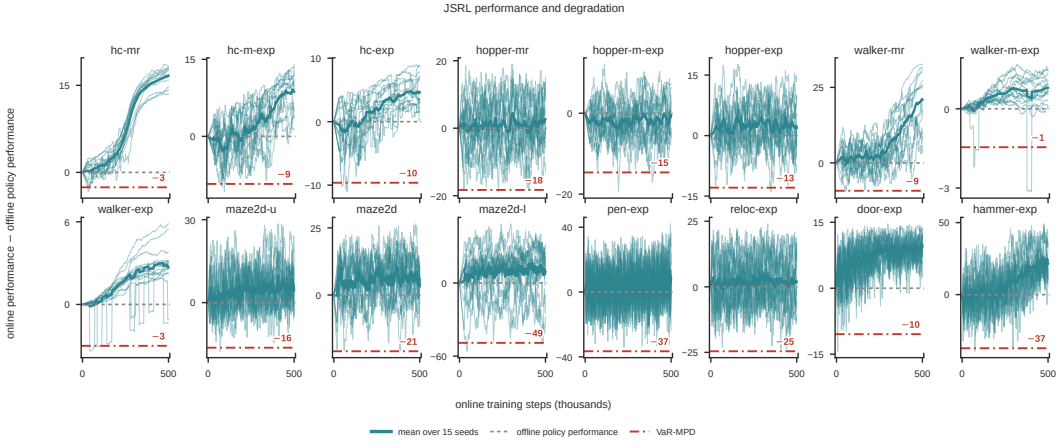


Figure 19: JSRL performance across all environments.

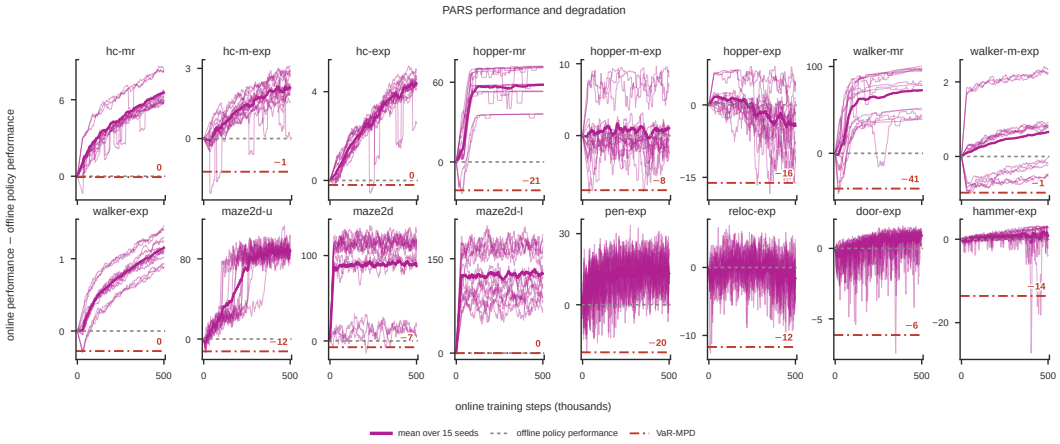


Figure 20: PARS performance across all environments.

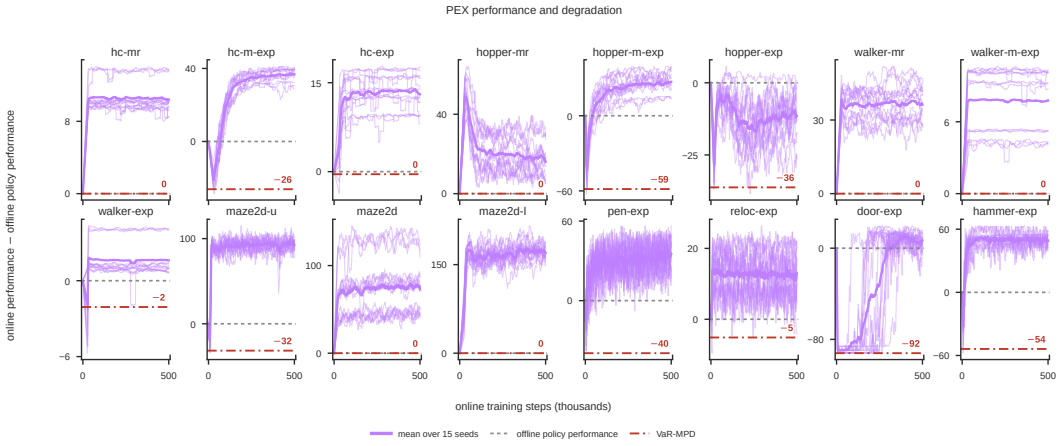


Figure 21: PEX performance across all environments.

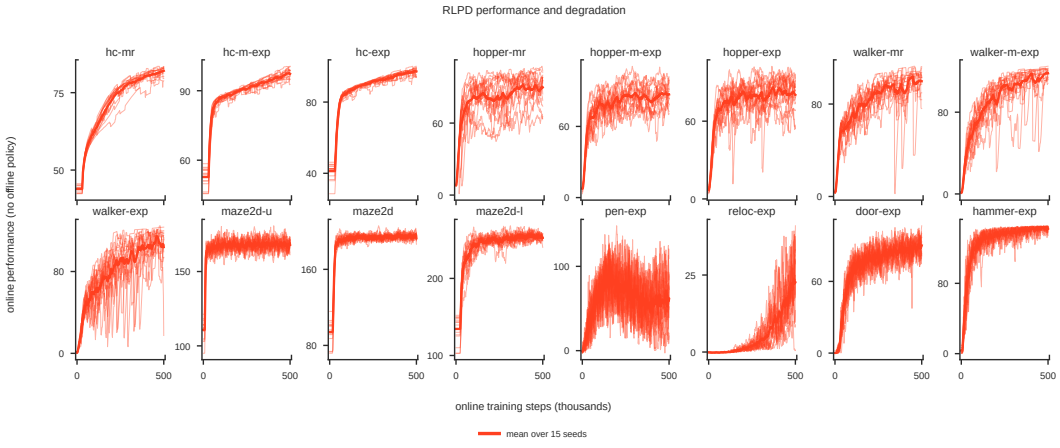


Figure 22: RLPD performance across all environments.

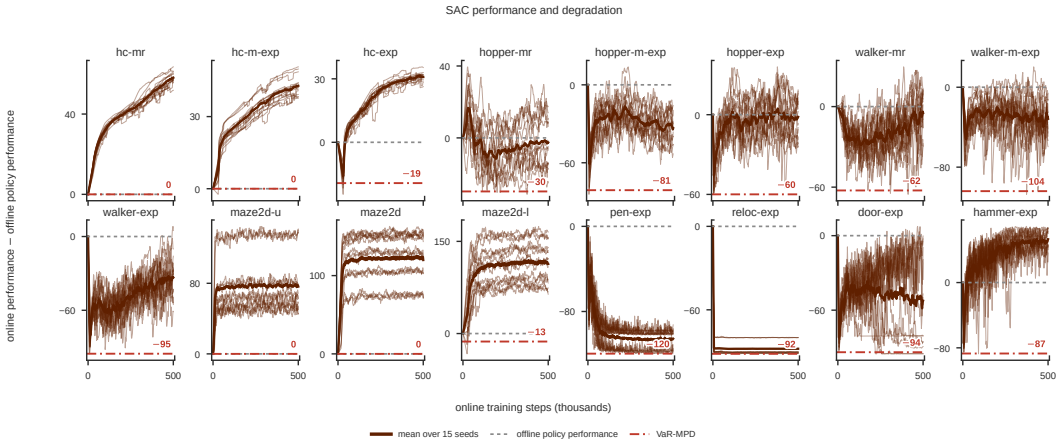


Figure 23: SAC performance across all environments.

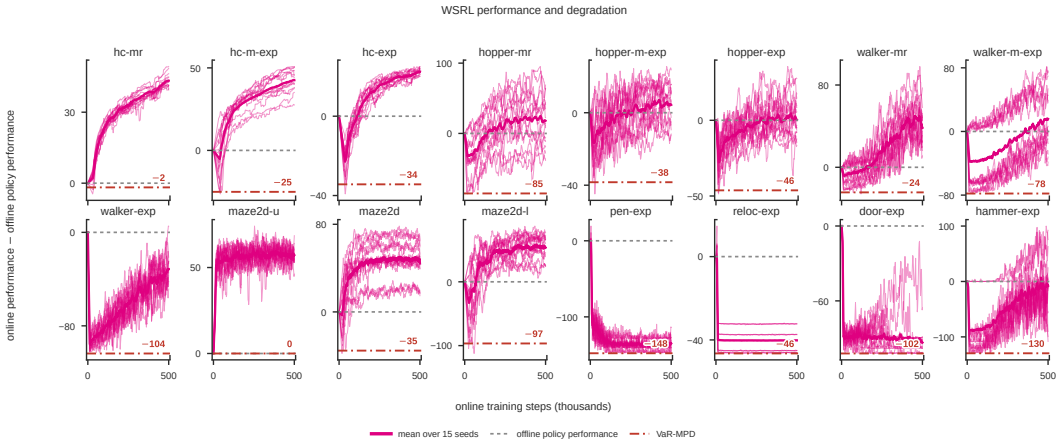


Figure 24: WSRL performance across all environments.

F INVESTIGATING WHY PERFORMANCE DEGRADATION HAPPENS

In this section, we provide some evidence that out-of-distribution issues are a primary culprit for performance degradation, and find several other hypotheses do not seem to be a key cause.

F.1 ENCOUNTERING NEW STATES AND ACTIONS DURING FINE-TUNING IS PROBLEMATIC

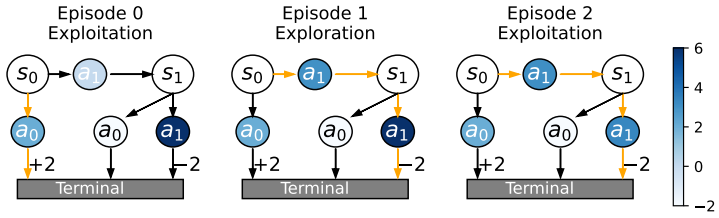


Figure 25: The effect of overestimation in states that are not covered by the dataset. Each subplot visualizes the state-action pairs in the environment at different time steps. The subtitle specifies whether the agent exploited at state s_0. Action values are represented by color intensity, with deeper colors indicating higher estimates. Yellow arrows refer to the path taken by the agent in that episode.

We hypothesize that a likely culprit for performance degradation is overestimation of unseen states. Conservative value estimates, given by algorithms like CQL and InAC, are only conservative for states in the dataset. Once online, however, the agent is likely to encounter new states and be potentially skewed by these overestimates.

To better understand this, let us consider a toy example in Fig. 25. Imagine in offline learning that s_1 is not in the dataset and we happen to have an overestimate for (s_1, a_1) (indicated by the dark blue), potentially due to an arbitrary initialization of the values for that unobserved state. Offline training does not change this arbitrary initial value, because the values for s_1 are never updated. During offline training, the agent learns that (s_0, a_0) has a reasonably high value, which is in fact the optimal choice. Once deployed, it will correctly take action a_0 from s_0 until it explores and takes action a_1. Once seeing state s_1, the agent will update the value of (s_0, a_1) by bootstrapping from the artificially high value (s_1, a_1). Now the agent will incorrectly think (s_0, a_1) is high-value, it will start going this suboptimal path, performance will degrade and it will take time for the agent to adjust those action-value estimates. Ultimately, it will relearn that a_0 is optimal from s_0, but in the interim we will see exactly the performance degradation we saw above.

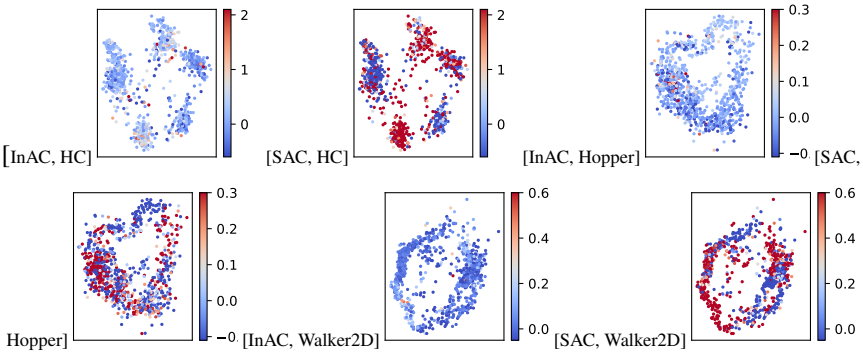


Figure 26: After 10,000 updates, SAC shifted toward selecting actions with higher initial value estimates. Each scatter represents a randomly sampled state from the offline dataset used for policy learning, with coordinates computed via PCA. The color of each scatter indicates the value difference between the updated policy and the initialized policy, as measured by the initial critic. A more intense red indicates a larger difference, meaning fine-tuning shifts the policy toward actions with higher initial value estimates. We set the color bar in the same environment to the same value range for clear visualization and shorten HalfCheetah to HC.

We also checked if SAC was directly suffering from overestimation for out-of-distribution actions, not just bootstrapping off of values for unvisited states. Let q_0 be the action-values learned offline

by InAC. We check how much SAC shifts action selection towards actions with high-value under q_0 for states in the offline dataset, but where InAC does not prefer those actions. If they are high-valued but not selected by InAC, it indicates they are out-of-distribution actions. We report this in Figure 26, where the more red there is in the scatterplot for SAC the more likely that it shifted to these out-of-distribution actions with arbitrarily high-value. This plot is generated by letting SAC fine-tune for 10k steps, which is generally a point where its performance has degraded, and then checking the value of SAC's policy according to q_0. Specifically, we subtract the value of SAC's policy under q_0 from the value of the offline policy under q_0. We provided InAC as a baseline, to show that unlike SAC, it generally did not shift value to these out-of-distribution actions.

F.2 Approaches that did not prevent degradation

In this section, we summarize the outcomes of experiments on two natural but failed ideas to prevent performance degradation.

Conservatism with ensembles did not prevent performance degradation. We explored the use of ensembles to provide conservative values for unseen states. The idea is that using the minimum of an ensemble of action-values alleviates issues with overestimation (Lee et al., 2021; Zhao et al., 2023). We found that the ensemble prevented degradation in some runs, but not in others, and generally negatively impacted learning during fine-tuning. The size of the ensemble had a big impact, in unintuitive ways. A small ensemble was typically better for fine-tuning, but suffered from the same degradation as not having an ensemble, but increasing the ensemble size did not predictably reduce degradation, and at times exacerbated it. Using some targeted experiments investigating the action-values themselves, we conclude that there are issues with action-value accuracy using an ensemble, and more research would need to be done to make ensembles effective for performance degradation.

Allowing a data collection phase did not prevent degradation. One hypothesis is that performance degradation is a forgetting issue (Wołczyk et al., 2024). This issue particularly arises when the online algorithm cannot leverage the offline data (e.g., it is not an off-policy algorithm), resulting in the need for other ways to remember what is learned in the offline policy. In our setting, we assume all the online algorithms can leverage the offline data, and initializing the online buffer with this data allows for rehearsal that alleviates any forgetting. Wołczyk et al. (2024) in fact show that this approach, which they call episodic memory, is effective to alleviate forgetting, if it is feasible to use. Nonetheless, one might imagine that the offline data is insufficient and that it is key to collect new data under the offline policy to reinforce what is learned in that policy.

We tested this hypothesis, by freezing the offline policy for a relatively large number of initial steps, storing and learning from this buffer before then allowing the policy to change. We found that this simply delayed performance degradation, and as soon as the policy started learning online (for example, with SAC), we observed performance degradation.