Bridging the Gap Between Offline and Online Reinforcement Learning Evaluation Methodologies

Anonymous authors
Paper under double-blind review

Abstract

Reinforcement learning (RL) has shown great promise with algorithms learning in environments with large state and action spaces purely from scalar reward signals. A crucial challenge for current deep RL algorithms is that they require a tremendous amount of environment interactions for learning. This can be infeasible in situations where such interactions are expensive, such as in robotics. Offline RL algorithms try to address this issue by bootstrapping the learning process from existing logged data without needing to interact with the environment from the very beginning. While online RL algorithms are typically evaluated as a function of the number of environment interactions, there isn't a single established protocol for evaluating offline RL methods. In this paper, we propose a sequential approach to evaluate offline RL algorithms as a function of the training set size and thus by their data efficiency. Sequential evaluation provides valuable insights into the data efficiency of the learning process and the robustness of algorithms to distribution changes in the dataset while also harmonizing the visualization of the offline and online learning phases. Our approach is generally applicable and easy to implement. We compare several existing offline RL algorithms using this approach and present insights from a variety of tasks and offline datasets.

1 Introduction

Reinforcement learning (RL) has shown great progress in recent years with algorithms learning to play highly complex games with large state and action spaces such as DoTA2 ($\approx 10^4$ valid actions) and StarCraft purely from a scalar reward signal (Berner et al., 2019; Vinyals et al., 2019). However, each of these breakthroughs required a tremendous amount of environment interactions, sometimes upwards of 40 years of accumulated experience in the game (Schrittwieser et al., 2019; Silver et al., 2016; 2018). This can be infeasible for applications where such interactions are expensive, for example robotics.

Offline RL methods tackle this problem by leveraging previously collected data to bootstrap the learning process towards a good policy. These methods can obtain behaviors that maximize rewards obtained from the system conditioned on a fixed dataset of experience. The existence of logged data from industrial applications provides ample data to train agents in simulation till they achieve good performance and then can be trained on real hardware. The downside of relying on offline data without any interactions with the system is that the behavior learned can be limited by the quality of data available (Levine et al., 2020).

Levine et al. (2020) point out that there is a lack of consensus in the offline RL community on evaluation protocols for these methods. The most widely used approach is to train for a fixed number of epochs on the offline dataset and report performance through the average return obtained over a number of episodes in the environment. However, this style of training does not provide insights into the data efficiency of offline RL algorithms. It also does not reveal how an algorithm reacts to changes in the dataset distribution, for example with better dataset-generating policies. In this paper, we propose to evaluate algorithms as a function of available data instead of just reporting final performance or plotting learning curves over a number of gradient steps. This approach allows us to study the sample efficiency and robustness of offline RL algorithms to distribution shifts. It also makes it easy to compare with online RL algorithms as well as

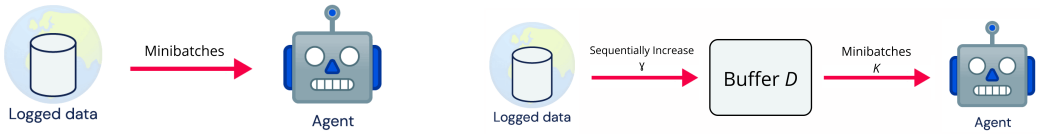
intuitively study online fine-tuning performance. We call this approach of evaluation Sequential Evaluation (SeqEval) and present settings where SeqEval can be integrated into the evaluation protocol of offline RL algorithms. We also propose a style of model card (Mitchell et al., 2018) that can be designed for offline RL algorithms to provide relevant context to practitioners about their sample efficiency to aid in algorithm selection.

2 Background and Related Work

In Reinforcement Learning (RL) we are concerned with learning a parametric policy $\pi_\theta(a|s)$ that maximizes the reward it obtains from an environment. The environment is described as a Markov Decision Process (MDP) using $< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} >$, where $\mathcal{S}$ represents the states of the environment, $\mathcal{A}$ represents the actions that can be taken, $\mathcal{R}$ is the set of rewards and $\mathcal{P}$ denotes the transition probability function $\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. The objective is to learn an optimal policy, denoted by π^*_θ, as $\pi^*_\theta = argmax_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$ where $\gamma \in [0, 1]$ is the discount factor and r_t is the per timestep reward. The Q value of a policy for a given state-action pair is the expected discounted return obtained from taking action a_t at state s_t and following the policy π afterwards. By estimating the Q value, we can implicitly represent a policy by always taking the action with the highest Q value. The Q value network is trained by minimizing the temporal difference (TD) error, defined as $(Q(s_t, a_t) - (r_t + \gamma \bar{Q}(s_{t+1}, a_{t+1})$. $\bar{Q}$ is called the target Q network and is usually a delayed copy of the Q network or a exponentially moving average of the Q network.

Offline RL. The simplest form of offline RL is behavior cloning (BC) which trains an agent to mimic the behavior present in the dataset, using the dataset actions as labels for supervised learning. However, offline datasets might have insufficient coverage of the states and operating conditions the agent will be exposed to. Hence BC agents tend to be fragile and can often perform poorly when deployed in the online environment. Therefore, offline RL algorithms have to balance two competing objectives, learning to generalize from the given dataset to novel conditions without deviating too far from the state space coverage in the dataset. The former can be enabled by algorithms that learn to identify and mimic portions of optimal behavior from different episodes that are suboptimal overall, as pointed out in Chen et al. (2021). The latter issue is addressed by constraining the agent's policy around behavior seen in the dataset, for example, through enforcing divergence penalties on the policy distribution (Peng et al., 2019; Nair et al., 2020). Another way of penalizing out of dataset predictions is by using a regularizer on the Q value, for example Conservative Q Learning (CQL) (Kumar et al., 2020) prevents actions that have low support in the data distribution from having high Q values. Decision Transformer (DT) (Chen et al., 2021) does not learn a Q estimate and instead formulates the RL task as a sequence modelling task conditioned on the return-to-go from a given state. The agent is then queried for the maximum return and predicts actions to generate a sequence that produces the queried return. This is a very brief overview of offline RL methods, and we direct readers to Levine et al. (2020) for a broader overview of the field.

Metrics and Objectives. The paradigm of *empirical risk minimization* (ERM) (Vapnik, 1991) is the prevailing training and evaluation protocol, both for supervised and unsupervised Deep Learning (DL). At its core, ERM assumes a fixed, stationary distribution and that we are given a set of (i.i.d.) data points for training and validation. Beyond ERM, and especially to accommodate non-stationary situations, different fields have converged to alternative evaluation metrics: In online learning, bandit research, and sequential decision making in general, the (cumulative) reward or the *regret* are of central interest. The regret is the cumulative loss accrued by an agent relative to an optimal agent when sequentially making decisions. For learning in situations where a single, potentially non-stationary sequence of observations is given, Minimum Description Length (MDL) (Rissanen, 1984) provides a theoretically sound approach to model evaluation. Multiple, subtly different formulations of the description length are in use, however, they are all closely related and asymptotically equivalent to *prequential MDL*, which is the cumulative log loss when sequentially predicting the next observation given all previous ones (Rissanen, 1987; Poland & Hutter, 2005). Similar approaches have been studied and are called the *prequential approach* (Dawid & Vovk, 1999) or simply *forward validation*. A common theme behind these metrics is that they consider the agents' ability to perform well, not only in the big-data regime, but also its generalization performance at the beginning, when only a few observations are available for learning. The MDL literature provides arguments and proofs



(a) Minibatch (b) Sequential Evaluation

Figure 1: Comparison of traditional training schemes with Sequential Evaluation

Algorithm 1 Algorithm for Sequential Evaluation in the offline setting.

Input: Algorithm A, Offline data $\mathcal{D} = \{s_t, a_t, r_t, s_{t+1}\}_{t=1}^{T}$, increment-size γ, gradient steps per increment K, evaluation frequency f_e
Replay-buffer $\mathcal{B} \leftarrow \{s_t, a_t, r_t, s_{t+1}\}_{t=1}^{T_0}$
$t \leftarrow T_0$
while $t < T$ **do**
  Update replay-buffer $\mathcal{B} \leftarrow \mathcal{B} \cup \{s_t, a_t, r_t, s_{t+1}\}_{t}^{t+\gamma}$
  Sample a training batch, ensure new data is included: batch $\sim \mathcal{B}$
  Perform training step with A on batch.
  $t \leftarrow t + \gamma$
  for $j = 1, \cdots, K$ **do**
    Sample a training batch $\sim \mathcal{B}$
    Perform training step with A on batch.
  end for
  if t % $f_e = 0$ **then**
    Evaluate A in the environment and log performance.
  end if
end while

why those models, that perform well in the small-data regime without sacrificing their big-data performance, are expected to generalize better to future data (Rathmanner & Hutter, 2011).

With these two aspects in mind, that a) RL deals with inherently non-stationary data, and b) that sample efficiency is a theoretically and practically desirable property, we propose to evaluate offline RL approaches by their data efficiency.

3 Sequential Evaluation of Offline RL Algorithms

As mentioned above, one approach for offline RL evaluation is to perform multiple epochs of training over the dataset. We contend that there are a few issues with this approach. Firstly, this approach does not provide much information about the sample efficiency of the algorithm since it is trained on all data at every epoch. This means that practitioners do not see how the algorithm can scale with the dataset size, or if it can achieve good performance even with small amounts of logged data. Furthermore, there can be distribution changes in the quality of the policy in the dataset, and evaluating as a function of epochs hides how algorithms react to these changes. Finally, there is a disconnection in the evaluation strategies of online and offline RL algorithms, which can make it difficult to compare algorithms realistically.

Instead of treating the dataset as a fixed entity, we propose that the portion of the dataset available to the agent changes over time and that the agent's performance is evaluated as a function of the available data. We term this as sequential evaluation of offline algorithms (SeqEval). This can be implemented by reusing any of the prevalent replay-buffer-based training schemes from online deep RL. Instead of extending the replay-buffer with sampled trajectories from the currently learned policy, we insert prerecorded offline RL data. That is, the offline dataset replaces the environment as the data generating function. We alternate between adding new samples to the buffer and performing gradient updates using mini-batches sampled from

the buffer. The number of samples added to the buffer at a time is denoted by γ and the number of gradient steps performed between each addition to the buffer is denoted by K. A concrete implementation of the approach is outlined in Alg. 1 and it is illustrated in Fig. 1.

This approach of evaluation addresses several of the issues with epoch-style training. By varying γ and K we can get information about the scaling performance of an algorithm with respect to dataset size, which tells us if data is the bottleneck for further improvements, and how quickly the algorithm can learn with limited data. We can visualize how the algorithm behaves with shifts in dataset quality directly from the performance curves. Online RL methods are evaluated as a function of the number of environment steps, which is a measure of the amount of data available to the agent. In the offline RL setting this would translate to evaluating the algorithm in terms of the amount of data it has access to in the replay buffer. We can also seamlessly evaluate the performance of the algorithm in online fine-tuning by adding samples from the environment once the entire offline dataset is added to the replay buffer. A benefit of the sequential approach is that it does not require a complete overhaul in codebases that follow existing training paradigms. For the baselines that we present in this paper, we were able to use the sequential evaluation approach with less than 10 lines of changes to the original codebases.

Implementation Details To ensure that the algorithm sees each data point in the dataset at least once, when a new batch of data is added to the buffer, the algorithm is trained on that sample of data once before K mini-batches are sampled from the buffer for training. If the batch added is smaller than the mini-batch size, then it can be made part of the next mini-batch that is trained on. In practice, we found that setting γ and K to 1 worked well in all datasets tested. This means that the x-axis of all plots directly corresponds to the number of samples available for training and the number of gradient updates performed. The changes made to the codebase of each algorithm are as follows: Each codebase had a notion of a replay buffer that was being sampled, and the only addition required here was a counter that kept track of up to which index in the buffer data points could be sampled from to create mini-batches. The counter was initialized to $T_0 = 5000$ so that there were some samples in the buffer at the start of training. The second change that needed to be made was changing the outer loop of training from epochs to the number of gradient updates and incrementing the buffer counter by γ every K update. This way, the amount of data the algorithm was trained on sequentially increased to the full dataset over the course of training.

4 Experiments

4.1 Baselines and Benchmarks

We evaluate several existing offline RL algorithms using the sequential approach, namely Implicit Q Learning (IQL) (Kostrikov et al., 2022), CQL (Kumar et al., 2020), TD3+BC (Fujimoto & Gu, 2021), AWAC (Nair et al., 2020), BCQ (Fujimoto et al., 2019), Decision Transformer (DT) (Chen et al., 2021) and Behavior Cloning (BC). These algorithms were evaluated on the D4RL benchmark (Fu et al., 2020), which consists of three environments: Halfcheetah-v2, Walker2d-v2 and Hopper-v2. For each environment, we evaluate four versions of the offline dataset: random, medium, medium-expert, and medium-replay. Random consists of 1M data points collected using a random policy. Medium contains 1M data points from a policy that was trained for one-third of the time needed for an expert policy, while medium-replay is the replay buffer that was used to train the policy. Medium-expert consists of a mix of 1M samples from the medium policy and 1M samples from the expert policy. These versions of the dataset are used to evaluate the performance of offline agents across a spectrum of dataset quality.

In addition to the D4RL benchmark, we created a dataset from the DeepMind Control Suite (DMC) (Tassa et al., 2018) environments following the same procedure as outlined by the authors of D4RL. We chose the DMC environments because of their high dimensional state space and action space. Specifically we chose cheetah run, walker run and finger turn hard and trained a Soft Actor Critic (SAC) (Haarnoja et al., 2018) agent on these environments for 1M steps. A medium and expert version of the dataset was created for each of the three environments using policies obtained after 500K and 1M steps respectively.

4.2 Experimental Settings

We study varied ways in which SeqEval can be integrated into offline RL training. The first setting is the "Standard" setting in which samples are added periodically to the buffer during training. The results of the Standard setting on a subset of datasets are given in Fig. 2 and the complete set of datasets, along with an experiment comparing curves with larger K are available in the appendix. Secondly, to highlight how SeqEval can visualize how the algorithm reacts to changing dataset quality during training, we create a "Distribution Shifts" setting where a mixed version of each environment is created. In this dataset, the first 33% of data comes from the random dataset, the next 33% from the medium dataset and the final 33% from the expert dataset. From the performance curves given in Fig. 5 we can see how each algorithm adapts to changes in the dataset distribution. We then show how SeqEval can be utilised in multi task offline RL in a similar fashion. Instead of providing the entire dataset of a new task to the agent at once, we periodically add new samples of the task to the agent over time. This lets us study the amount of samples required for the agent to transfer to the new task. Finally we also show how SeqEval supports seamless integration of online fine-tuning experiments into performance curves. In this setting, once the entire offline dataset is added to the replay buffer, the agent is allowed to interact with the online simulator for a fixed number of steps (500k steps in our experiments). Since the curves are a function of data samples, we can continue evaluating performance as before. The results on a subset of datasets are given in Fig. 6, and curves for all datasets are available in the appendix.

For each dataset, we train algorithms following Alg. 1, initializing the replay buffer with 5000 data points at the start of training. We set γ and K each to 1, that is, there is one gradient update performed on a batch of data sampled from the environment every time a data point is added to the buffer. The x-axis in the performance curves represents the amount of data in the replay buffer. In each plot, we also include the performance of the policy that generated the dataset as a baseline, which provides context for how much information each algorithm was able to extract from the dataset. This baseline is given as a horizontal dotted line.

4.3 Model Cards

Since SeqEval evaluates algorithms as a function of data, it opens up the possibility of quantitatively analysing how algorithms scale with data. We propose a few evaluation criteria in addition to the training curves. The first one is Perf@50%, which measures the aggregate performance of algorithms when half the dataset is available to it for training. To further analyze the role of data scaling, we studied two measures, Perf@50% normalised with respect to performance obtained with the whole dataset (Perf@100%) and the difference between Perf@100% and Perf@50%. These three metrics give end users valuable insights that can aid in offline algorithm selection. They can be presented as part of an overall model card (Mitchell et al., 2018) available to the user.

4.4 Results

Standard We present the results of the Standard setting in Figs. 2 and 3. One striking observation from Fig. 2 is how quickly the algorithms converge to a given performance level and then stagnate. This is most evident in the medium version of each environment. With less then 300K data points in the buffer, each algorithm stagnates and does not improve in performance even after another 500K points are added. That is, there are diminishing returns from adding data beyond 500K points to the buffer. This highlights that most of the tested algorithms are not very data-hungry. That is, they do not require a large data store to reach good performance, which is beneficial when they need to be employed in practical applications. The experiment highlights that the chosen datasets might lack diversity in collected experience since most algorithms appear to need only a fraction of it to attain good performance. This phenomenon is made more stark in Fig. 4. Most algorithms reach within 90% of final performance with just 50% of the data. In other words, doubling the data in the D4RL benchmarks provides an aggregated uplift of 10% at best. This indicates that more effort could be put into designing a harder benchmark. The effect is less pronounced in the DMC datasets, where the algorithms continue learning till about 50% of the dataset is added and then learning begins to stagnate.

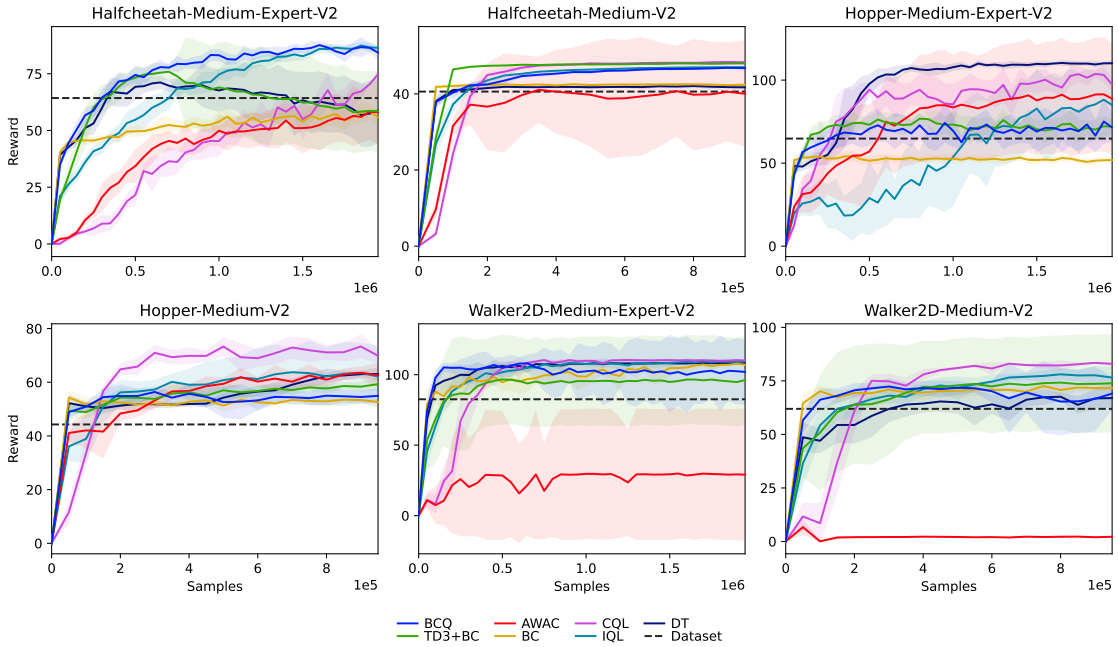


Figure 2: Performance curves on the D4RL benchmark as a function of data points seen. Shaded regions represent standard deviation across 5 seeds.

In a majority of the datasets tested on D4RL, CQL reaches better performance than other methods earlier in training and consistently stays at that level as more data is added. TD3+BC exhibits an initial steep rise in performance but levels out at a lower score overall, or in the case of Halfcheetah-medium-expert, degrades in performance as training progresses.

Distribution Shifts The learning curves for the "mixed" dataset in the D4RL benchmark and DMC are given in Fig. 5. In the mixed dataset experiment, most of the algorithms do adapt to the new data continually improving each time there is a shift in the dataset quality. However rarely are they able to do so in every single environment. While CQL adapts quickly in both HalfCheetah and Walker2d, it fails to learn at all in Hopper. TD3+BC outperforms all other methods in HalfCheetah, but is the worse performing in Walker2d. DT is able to consistently adapt to new data in each environment. A surprising result is how well BC performs in each environment, with BC nearly being the second best performing algorithm in all environments. We hypothesise that since there is good coverage of the state spaces and optimal behavior in the dataset, behavior cloning is able to learn effectively. It does not have to "stitch" behavior from different episodes and instead just need to replicate the behavior in the dataset. This is in line with the findings of Chen et al. (2021) which found that 10%BC, which trained BC on only the top 10% percentile of episodes (based on episodic return), was competitive with specialised offline algorithms on the D4RL benchmark.

Online Finetuning In online fine-tuning, shown in Fig. 6 and Table 3, CQL reaches higher scores compared to the other algorithms showing the versatility of CQL in handling both offline datasets and online interactions with the system. While in the random dataset there is a jump in performance after online finetuning, there aren't large gains in other datasets. This is evident from Fig. 4c, where we see that in general there are modest gains from online finetuning in these algorithms.

5 Conclusion

In this paper, we propose a sequential style of evaluation for offline RL methods so that algorithms are evaluated as a function of data rather than compute or gradient steps. In this style of evaluation, data is added sequentially to a replay buffer over time, and mini-batches are sampled from this buffer for training.

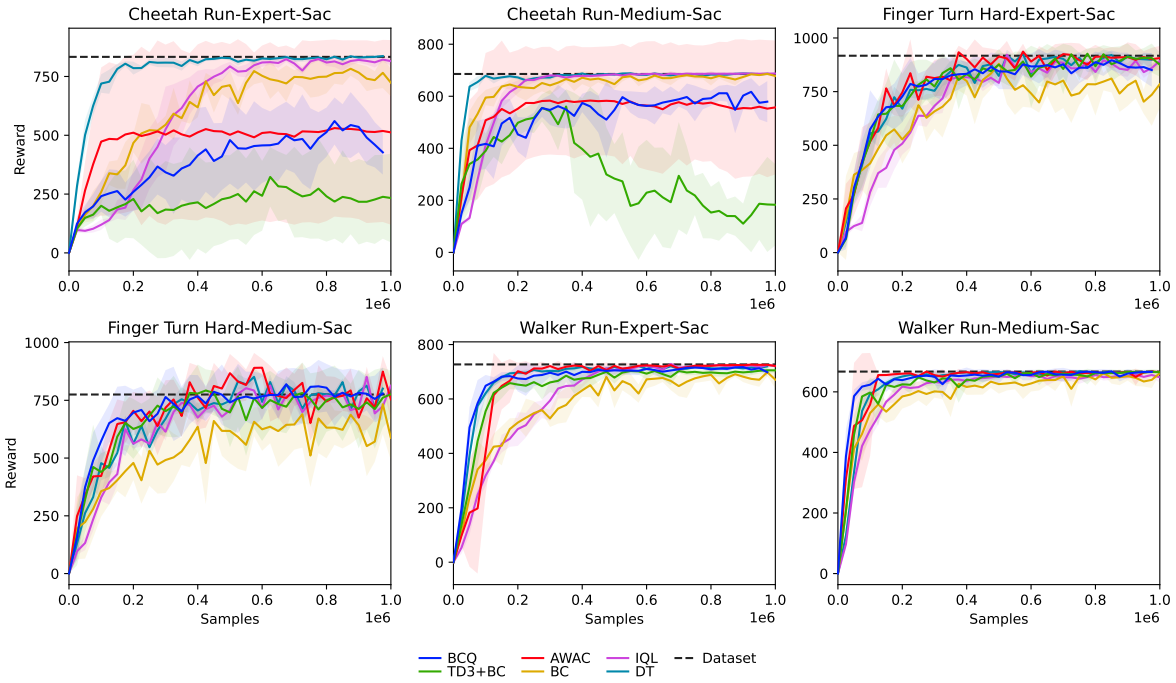
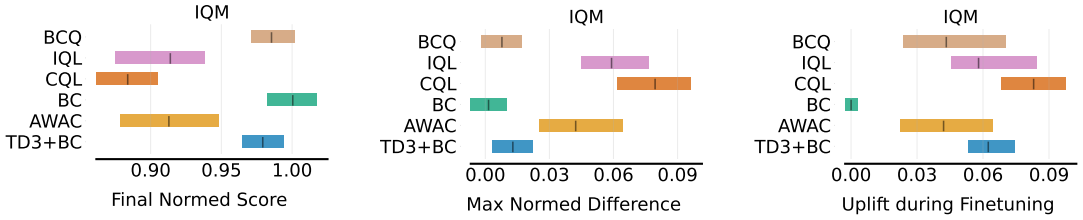


Figure 3: Performance curves on the DMC Dataset as a function of data points seen. Shaded regions represent standard deviation across 5 seeds.



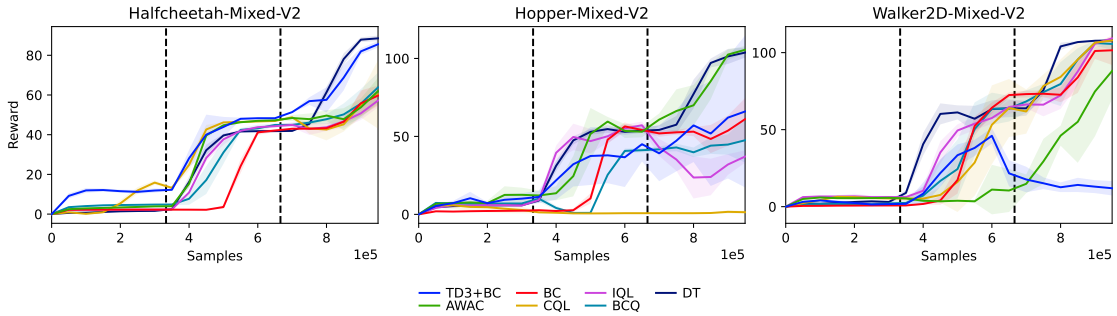
(a) $\frac{Perf@50\%}{Perf@100\%}$ (b) Perf@100% − Perf@50% (c) Finetuning Uplift

Figure 4: Analysis of data scaling in the D4RL benchmark. Perf@X% refers to performance when the algorithm has seen $X\%$ of the dataset. a) and b): Standard Setting c) Online Finetuning setting.

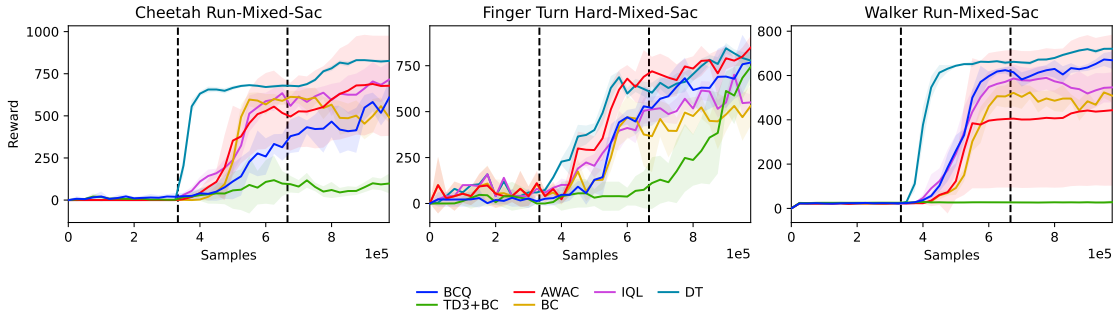
This is analogous to online training to deep RL and allows us to measure the data scaling and robustness of offline algorithms simultaneously from the training curves. We compared several existing offline methods using sequential evaluation and showed how their training curves allow for algorithm selection depending on data efficiency or performance. We believe that sequential evaluation holds promise as an established method of evaluation for the offline RL community. Future work in this domain could explore the effect of γ and K on algorithms and their ramifications on sample efficiency. While we have used the evaluation episodic return as the performance metric in our experiments, it is not necessary to do so. For example, if there are environments where a simulator is not available for periodic cheap evaluation, off-policy evaluation metrics Thomas et al. (2015); Wang et al. (2020) can be used.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.



(a) D4RL



(b) DMC

Figure 5: Performance curves for Mixed datasets with varying dataset quality. Dotted line indicates where there is a change in the dataset generating policy distribution. Shaded regions represent standard deviation across 5 seeds.

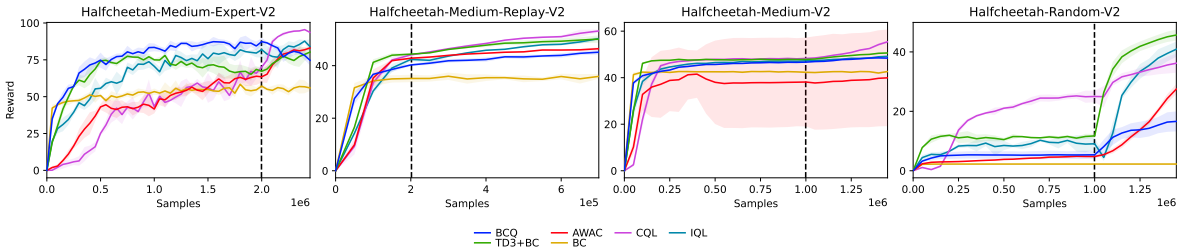


Figure 6: Performance curves for online fine-tuning. Each algorithm is given 500k steps in the simulator after sequential evaluation of the offline dataset. Dotted line indicates where online fine-tuning begins. Shaded regions represent standard deviation across 3 seeds.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*, 2021.

A. Philip Dawid and Vladimir G. Vovk. Prequential probability: principles and properties. *Bernoulli*, 5(1): 125–162, 1999.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pp. 2052–2062, 2019.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1856–1865. PMLR, 2018. URL `http://proceedings.mlr.press/v80/haarnoja18b.html`.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL `https://openreview.net/forum?id=68n2s9ZJWF8`.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/0d2b2061826a5df3221116a5085a6052-Abstract.html`.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv: Arxiv-2005.01643*, 2020.

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, B. Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. *FAT*, 2018. doi: 10.1145/3287560.3287596.

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

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *CoRR*, abs/1910.00177, 2019. URL https://arxiv.org/abs/1910.00177.

Jan Poland and Marcus Hutter. Asymptotics of discrete MDL for online prediction. *IEEE Transactions on Information Theory*, 51(11):3780–3795, 2005.

Samuel Rathmanner and Marcus Hutter. A philosophical treatise of universal induction. *Entropy*, 13(6): 1076–1136, 2011. ISSN 1099-4300. doi: 10.3390/e13061076.

J Rissanen. Universal coding, information, prediction, and estimation. *IEEE Trans. Inf. Theory*, 30(4): 629–636, July 1984.

Jorma Rissanen. Stochastic complexity. *Journal of the Royal Statistical Society: Series B (Methodological)*, 49(3):223–239, 1987.

Julian Schrittwieser, Ioannis Antonoglou, T. Hubert, K. Simonyan, L. Sifre, Simon Schmitt, A. Guez, Edward Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 2019. doi: 10.1038/s41586-020-03051-4.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. doi: 10.1038/nature16961. URL https://doi.org/10.1038/nature16961.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362:1140–1144, 2018. doi: 10.1126/science.aar6404. URL https://syndication.highwire.org/content/doi/10.1126/science.aar6404.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite. *arXiv preprint arXiv: Arxiv-1801.00690*, 2018.

Philip S. Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High confidence off-policy evaluation. *AAAI*, 2015. URL https://dl.acm.org/doi/10.5555/2888116.2888134.

V Vapnik. Principles of risk minimization for learning theory. In *Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS'91, pp. 831–838, San Francisco, CA, USA, December 1991. Morgan Kaufmann Publishers Inc.

Oriol Vinyals, Igor Babuschkin na, Wojciech M. Czarnecki na, Michaël Mathieu na, Andrew Dudzik na, Junyoung Chung na, David H. Choi na, Richard Powell na, Timo Ewalds na, Petko Georgiev na, Junhyuk Oh na, Dan Horgan na, Manuel Kroiss na, Ivo Danihelka na, Aja Huang na, Laurent Sifre na, Trevor Cai na, John P. Agapiou na, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps na, and David Silver na. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 2019. doi: 10.1038/s41586-019-1724-z. URL https://doi.org/10.1038/s41586-019-1724-z.

Jie Wang, Rui Gao, and Hongyuan Zha. Reliable off-policy evaluation for reinforcement learning. *arXiv preprint arXiv: Arxiv-2011.04102*, 2020.

A Description of datasets

The D4RL locomotion benchmark consists of three environments with varying data quality. The size of each version and the performance of the policy that generated it is given in Table 1.

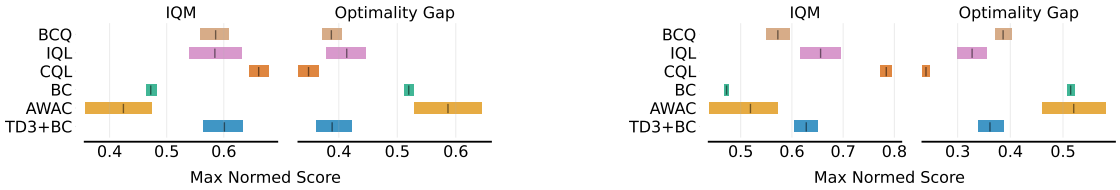
Table 1: Dataset sizes in D4RL

Dataset	Size	Average score in Dataset
halfcheetah-random-v2	999000	-0.07
halfcheetah-medium-v2	999000	40.60
halfcheetah-medium-replay-v2	201798	27.03
halfcheetah-medium-expert-v2	1998000	64.29
halfcheetah-expert-v2	999000	87.91
walker2d-random-v2	999999	0.01
walker2d-medium-v2	999322	61.94
walker2d-medium-replay-v2	301698	14.81
walker2d-medium-expert-v2	1998318	82.55
walker2d-expert-v2	999000	106.90
hopper-random-v2	999999	1.19
hopper-medium-v2	999998	44.28
hopper-medium-replay-v2	401598	14.97
hopper-medium-expert-v2	1998966	64.78
hopper-expert-v2	999061	108.24

B Performance on D4RL Benchmark

We present complete training curves on all twelve datasets that were used in Fig. 8 and final performance in Table 2. In addition to the curves, we compare the algorithms at the end of training with scores aggregated across environments. This is done using the rliable (Agarwal et al., 2021) library to plot interval estimates of normalized performance measures such as median, mean, interquartile mean (IQM) and optimality gap. The optimality gap is a measure of how far an algorithm is from optimal performance aggregated across environments. So, lower values are better. The scores in each dataset are normalized with respect to the maximum score, which is 100. These results are given in Fig. 10. In both the performance curves and aggregated scores we can see that CQL outperforms other tested methods by a clear margin. A curious phenomenon observed was that AWAC (Nair et al., 2020) is unable to learn at all in the Walker2d environment with either the medium-expert or the medium version achieving very low rewards. As a sanity check we set γ to the size of the dataset and reran experiments and the method achieved results similar to those reported in the paper, indicating it was not an implementation problem. This is surprising since AWAC is proposed as an algorithm that can work for online fine-tuning following offline pretraining, but among all the methods tested, it had drastic changes in performance when using sequential evaluation. The final performance in the online fine-tuning task and the mixed version of the environment is also given in Tables 3 and 4. The Perf@50% and Perf@100% results are given in Fig. 7

Moreover, Fig. 11 show how the algorithms performed when K was set to 2. This experiment studied if we had not trained the methods for enough gradient steps and if additional performance could be extracted from the data. However, performance remained essentially the same or even degraded in some instances, showing that this was not the case.



(a) Perf@50%

(b) Perf@100%

Figure 7: Comparison of performance when a) half the dataset is available b) full dataset is available

Table 2: Performance of each algorithm on the D4RL Benchmark

Dataset	BCQ	TD3+BC	AWAC	BC	CQL	DT	IQL
halfcheetah-medium-expert-v2	92.25	6.83	59.04	63.44	83.45	0.0	86.0
halfcheetah-medium-replay-v2	37.54	44.26	43.37	36.3	43.54	0.0	41.97
halfcheetah-medium-v2	45.47	48.45	46.58	43.12	49.13	42.65	46.96
hopper-medium-expert-v2	108.8	69.92	111.88	44.53	111.0	0.0	49.18
hopper-medium-replay-v2	23.9	63.04	65.47	14.02	88.51	0.0	69.09
hopper-medium-v2	53.39	49.77	52.8	55.84	70.53	62.06	67.49
walker2d-medium-expert-v2	110.36	108.33	1.98	107.58	109.98	0.0	98.51
walker2d-medium-replay-v2	59.94	76.8	82.54	24.63	73.31	0.0	63.12
walker2d-medium-v2	76.85	83.4	1.76	78.78	83.16	65.79	80.85

Table 3: Performance of each algorithm in the online fine-tuning task on the D4RL Benchmark

Dataset	TD3+BC	AWAC	BC	CQL	IQL
finetune-halfcheetah-medium-expert-v2	74.36	83.6	61.77	96.72	87.74
finetune-halfcheetah-medium-replay-v2	50.52	46.87	28.66	53.25	49.53
finetune-halfcheetah-medium-v2	51.9	52.89	42.86	55.29	49.24
finetune-halfcheetah-random-v2	48.53	30.67	2.26	34.17	50.87
finetune-hopper-medium-expert-v2	112.71	111.9	48.01	100.25	110.66
finetune-hopper-medium-replay-v2	92.56	65.56	13.19	102.78	100.38
finetune-hopper-medium-v2	63.03	45.45	58.92	94.31	73.59
finetune-hopper-random-v2	9.85	8.78	2.47	4.56	12.05
finetune-walker2d-medium-expert-v2	107.77	1.41	108.35	110.73	110.81
finetune-walker2d-medium-replay-v2	77.78	94.11	5.61	89.77	95.71
finetune-walker2d-medium-v2	85.01	21.32	66.47	83.7	84.24
finetune-walker2d-random-v2	8.6	1.67	0.5	0.33	10.56

Table 4: Performance of each algorithm in the mixed version of the D4RL Benchmark

Dataset	BCQ	TD3+BC	AWAC	BC	CQL	DT	IQL
halfcheetah-mixed-v2	66.98	80.65	29.55	57.38	93.47	81.51	68.04
hopper-mixed-v2	55.29	112.42	110.36	86.67	0.75	111.74	51.37
walker2d-mixed-v2	102.77	8.15	98.36	108.79	109.71	107.63	109.83

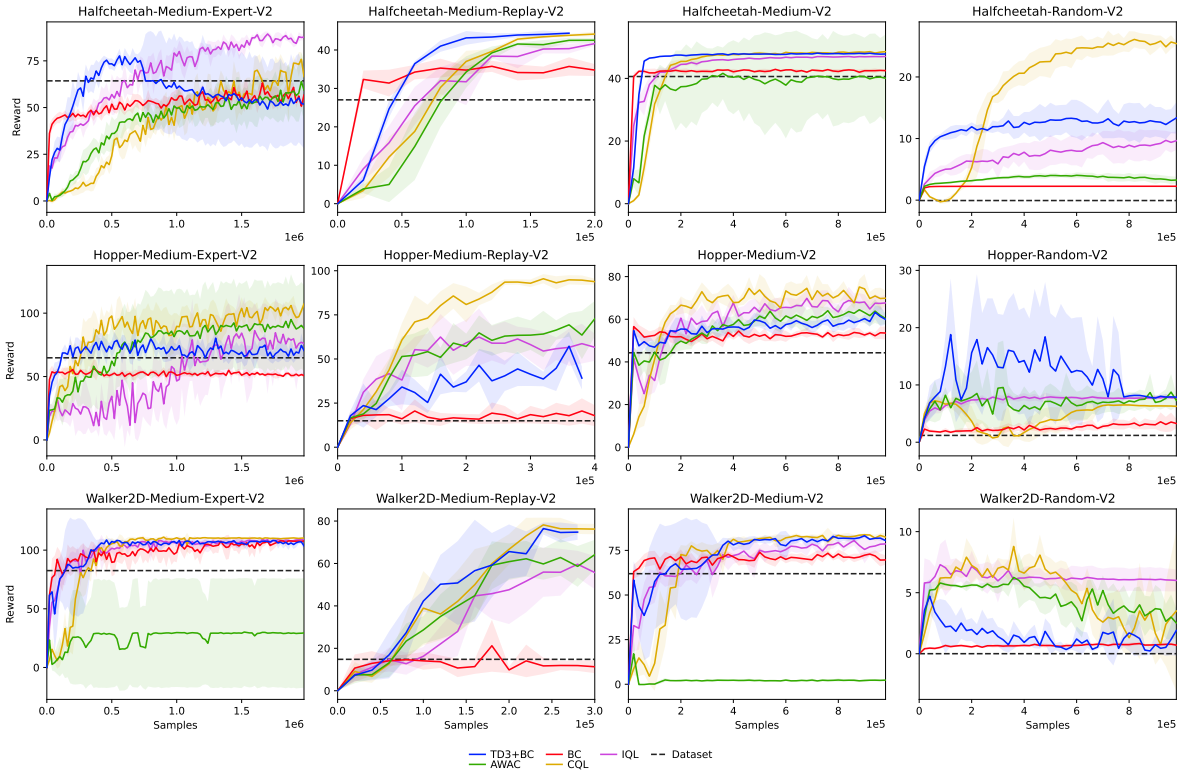


Figure 8: Performance curves on the D4RL benchmark of offline RL algorithms as a function of data points seen. Shaded regions represent standard deviation across 5 seeds.

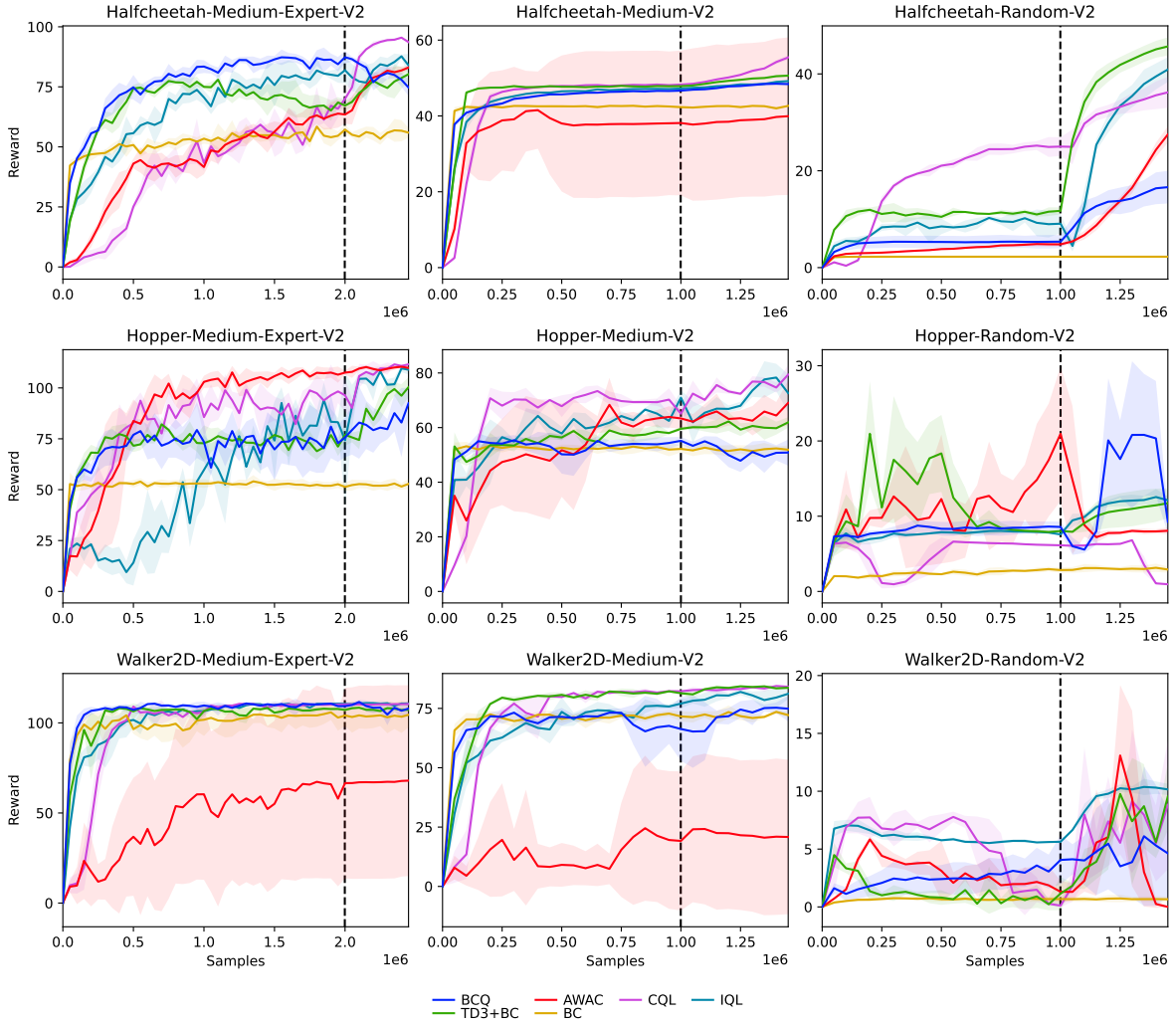


Figure 9: Performance curves for online fine-tuning. Each algorithm is given 500k steps in the simulator after sequential evaluation of the offline dataset. Dotted line indicates where online fine-tuning begins. Shaded regions represent standard deviation across 3 seeds.

(a) Only offline Dataset

(b) Online fine-tuning

Figure 10: Performance aggregated across environments using rliable. For IQM higher is better, while for Optimality gap, lower is better

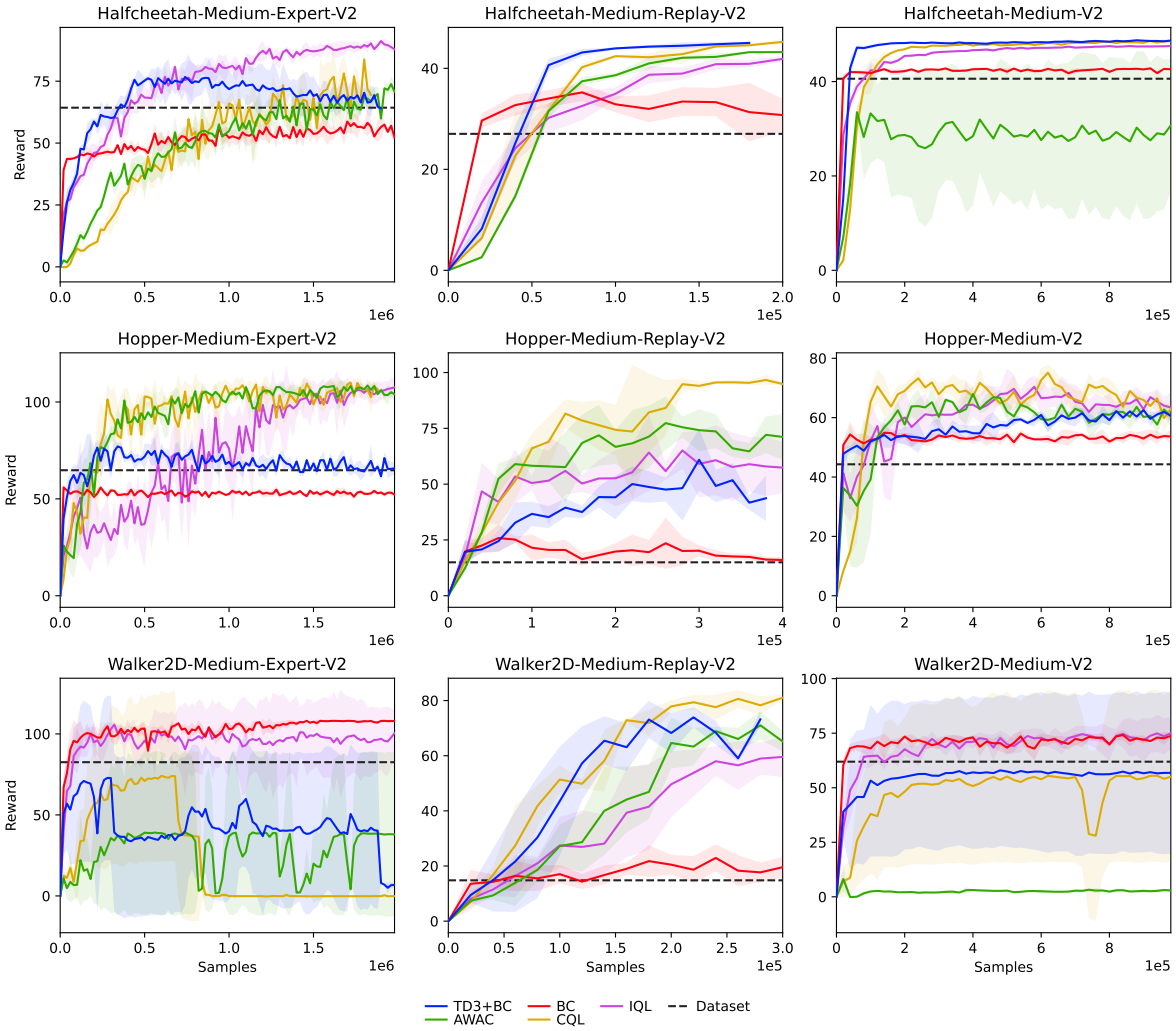


Figure 11: Performance curves on the D4RL benchmark with K increased to 2. Shaded regions represent standard deviation across 3 seeds.