Multi-task Offline Reinforcement Learning with CQL

A study on how dataset size and diversity increase generalization performance

Laimonas Lipinskas
Supervisors: Matthijs Spaan, Max Weltevrede
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Laimonas Lipinskas
Final project course: CSE3000 Research Project
Thesis committee: Matthijs Spaan, Max Weltevrede, Elena Congeduti

Abstract

Reinforcement learning (RL) is a type of machine learning where a model learns by making an observation of the current state it is in, picking out an action to execute, and observing the reward of said action, after which it receives the next state and repeats the cycle until it reaches its goal. The traditional online training approach allows the agent to directly interact with the live environment, but that is not always possible due to the live environment possibly being too dangerous or costly to train in. In cases like these offline training, which instead trains the agent on already pre-collected datasets of previously mentioned interactions and tries to learn a better policy than the one used for collection, offers a viable alternative by employing Q-Learning methods like CQL. However, prior studies, such as Mediratta et al. [11], have suggested that Behavior Cloning (BC), a type of imitation cloning, may outperform modern offline RL methods in the multi-task setting, where model generalization is tested on new or similar tasks rather than the ones trained on. Considering these results, it begs the question whether it is worthwhile to employ modern Q-Learning methods designed to derive a better policy than the one used to collect the data, especially when they are unable to outperform standard imitation learning.

This study seeks to reproduce and extend these findings within a custom environment. The results reveal that, contrary to the aforementioned report, BC does not consistently outperform CQL. Both machine learning methods exhibit comparable performance across datasets varying in diversity and size. Additionally, incorporating more diverse data significantly enhances generalization performance.

1 Introduction

Reinforcement learning (RL) has demonstrated effectiveness in various fields, ranging from simulated autonomous vehicle control [6] to applications in finance [4]. Common training practices often involve training a model in an online setting, meaning the agent can interact directly with the environment by first making an observation, then choosing an action, and finally learning from the reward. However, this approach is sometimes not possible due to limited access to live environments or the high risks associated with live training. In such a case, offline training can be a suitable alternative. The idea behind this type of learning is that the agent, instead of interacting with the environment directly, learns from a dataset that is made up of already collected experiences gathered by some data collection policy. This type of learning aims to not only copy the behavior of the collection policy but also improve on it. Recent advances in Q-Learning methods, such as Conservative Q-Learning (CQL) [9] and Implicit Q-Learning (IQL) [8], have shown improvements in offline RL by addressing distributional shift, a significant challenge rising from mismatches between training data and real-life application environments.

Although training on a single task and perfecting it can be beneficial, generalization is often a critical attribute for RL models as it helps models adapt to real-world applications more effectively. The paper by Mediratta et al [11] reveals that modern offline RL methods do not outperform Behavior Cloning (BC) in a multi-task setting where the agent trains on several tasks and has to learn a policy that can adapt to similar but new tasks. Their experiment shows that imitation learning provides the best performance when trained on datasets composed of optimal actions, which is understandable, as cloning behavior is the idea behind the method. However, BC also outperforms other methods for datasets containing more diverse actions, where theoretically, Q-Learning methods should be able to find a better policy than the one used for data collection. The main issue highlighted by these findings is that if methods designed to improve upon the collection policy cannot even surpass the results of simple imitation, then there may be little reason to use them.

This study aims to reproduce some results from the multi-task setting experiments shown in the paper by Mediratta et al. [11]but within a slightly modified Four Room environment provided by Minigrid [3]. The findings show that BC does not outperform CQL as clearly as reported in the aforementioned paper, neither when trained on optimal datasets, nor when trained on the suboptimal and newly added random walk ones, with performance largely being equal between the machine learning methods. Additionally, this study demonstrates that both adding more diverse data and increasing dataset size yields non-marginal performance improvements when evaluating, helping to reduce the generalization gap between training and testing rewards in the custom environment.

2 Background

This section provides a consice introduction to RL, followed by a short overview of offline RL and mentioning some of its advantages and disadvantages. Additionally, it introduces the machine learning methods used in this study and discusses the concept of reachability, which will be used in evaluating these methods later on.

2.1 Reinforcement Learning and Q-Learning

Reinforcement learning is a machine learning approach that optimizes the Markov Decision Process (MDP) to find an optimal solution to a given problem. An MDP is typically made up of six components: $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, p_0, \gamma)$. Here, $\mathcal{S}$ represents the set of all possible states an agent can be in, and $\mathcal{A}$ is the set of all possible actions the agent can take. The transition function $\mathcal{T}(s, a, s_{t+1}) = P(s_{t+1}|s, a)$ defines the probability of transitioning to state s_{t+1} given the current state s and action a. The reward function $\mathcal{R}(s, a)$ returns the reward received after taking action a in state s. The initial state distribution p_0 specifies the probability distribution over starting states, and γ is a discount factor that tweaks the importance of immediate and future rewards.

The goal in RL is to find a policy $\pi(a|s)$ that maximizes the expected cumulative future rewards. This can be mathematically expressed as:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \right] \tag{1}$$

where the expectation $\mathbb{E}_{\pi}$ is taken over the trajectories generated by following the policy π [2].

Q-Learning is a model-free RL algorithm that aims to learn the quality of actions, denoted as Q-values, for each state-action pair. It seeks to find the optimal policy by iteratively updating Q-values using the following update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{2}$$

Here, α is the learning rate, r is the reward received after taking action a_t in state s_t, s_{t+1} is the next state, and a' are actions that are possible to take from state s_{t+1}. By repeatedly updating Q-values through exploration and exploitation, Q-Learning converges to the optimal policy that maximizes cumulative rewards.

2.2 Offline Reinforcement Learning and The Problem of Distributional Shift

There are two primary types of RL: online and offline. In online RL, an agent interacts directly with the environment to test actions and observe their rewards. In contrast, offline

RL exclusively uses a static dataset $D = \{(s_t, a_t, s_{t+1}, r_t)\}$, collected by a predetermined policy π_β, to determine an optimal policy without further interaction with the environment [10].

Offline RL is often used when training an agent in an online setting is too costly, or as a pre-training step before fine-tuning to reach an acceptable performance that can be tweaked further in a live environment. An example of costly training is developing a model to prescribe accurate treatment plans for patients, where the state would be the patient's complaints and the action would be prescribing some treatment plan. While the idea of a scalable AI system addressing individual healthcare needs is appealing, training such a model online is dangerous, as incorrect treatment plans could have severe consequences on human life. Offline training is perfect for this scenario because the model can learn from previous doctors' treatments and outcomes without taking the risks in order to improve. In general, the offline approach provides the advantages of being able to reuse existing datasets and the ability to rapidly train on a computer without the need for any interaction in a real-life environment.

However, the accuracy of such training can be hindered by distributional shift, which occurs when the distribution of the gathered dataset's policy differs from that of the learned agent's policy. For example, consider a healthcare model trained mainly on data from young patients; this model might recommend inappropriate treatments for elderly patients due to differences in medication suitability. Another example could be a shift in actions, a model trained mostly on conservative treatments might mistakenly conclude that radical treatments are far better since it was able to find a scenario when it worked once or twice, whereas in reality the failures of the radical methods were just not documented well enough in the dataset.

2.3 Behavior Cloning

BC is a machine learning method that aims to imitate the actions provided by an expert or another data collection policy. In simple mathematical terms, the objective of the method is to minimize the objective function, where d^* is a dataset of state-action pairs (s, a), and π^* is the policy it wishes to imitate by approximating it via maximum-likelihood optimization. This can be written as a Function 3, where (s, a) are state-action pairs, $s \sim d^*$, and $a \sim \pi^*(s)$ [12].

$$J_{\mathrm{BC}}(\pi^) \coloneqq \mathbb{E}_{(s,a)\sim(d^*,\pi^*)}[-\log \pi(a \mid s)] \tag{3}$$

An imitation learning technique is well-suited for offline settings, as it can be easily trained using an existing dataset, but it does nothing in order to counter the problem of distributional shift. The idea behind using this method as a baseline is that if more complex RL methods cannot outperform simple imitation, there is little justification for using them to achieve an optimized policy that might surpass one obtained through straightforward behavioral cloning from the original dataset.

2.4 Conservative Q-Learning

In standard offline Q-Learning, underrepresented or missing actions in the dataset may lead to the Q-function overestimating the value of some states. This issue could be addressed in an online setting by selecting the action and evaluating its value, but this is not possible in the offline setting. Consequently, these methods might rely on out-of-distribution values, significantly overestimating their performance 1.

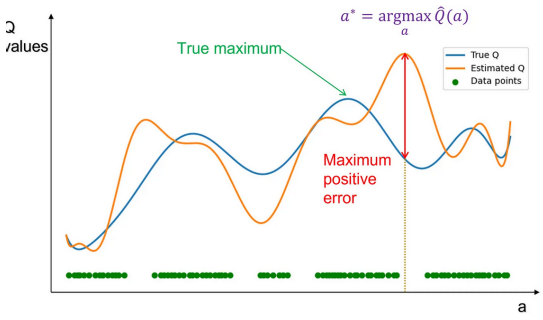


Figure 1: Q-function overestimation on low-density dataset points. Image is taken from [5]

CQL is a method developed to try and mitigate this problem. The general idea behind it is to try to adjust the Q-Function in such a way that the Q-values for actions seen in the dataset are pushed up, while the Q-values for actions not or rarely seen in the dataset are pushed down. This helps create a more conservative estimate, which can prevent the learned policy from making risky decisions based on overestimated action values that were not well-represented in the training data.

In the Q-value update function 4, taken from Kumar et al. [9], the blue operation aims to create a conservative bound for the value function of the policy being optimized. It computes the minimum over all Q-values, where the left expectation of the blue operation decreases Q-values for all actions, and the right expectation increases those observed in the dataset collected by policy π_β. For large Q-Values that are both in the dataset and the learned policy, this tends to even out, leaving just the green operation, which is the standard Bellman error objective to take over. Consequently, the values that are not in the dataset are only pushed down, thus giving more importance to the values that are sampled from the dataset.

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \alpha \cdot \left(\mathbb{E}_{\mathbf{s}\sim d^{\pi_\beta}(\mathbf{s}),\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] - \mathbb{E}_{\mathbf{s}\sim d^{\pi_\beta}(\mathbf{s}),\mathbf{a}\sim\pi_\beta(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]\right) + \frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[\left(Q(\mathbf{s},\mathbf{a}) - \mathcal{B}^\pi\hat{Q}^k(\mathbf{s},\mathbf{a})\right)^2\right]. \tag{4}$$

2.5 Reachable and Unreachable States

A state s_b is considered reachable if there exists a sequence of actions starting from some initial state s_a that can lead to s_b. Conversely, an unreachable state is one where no sequence of actions, starting from any state, can reach it. In the context of Figure 2, this includes everything outside the depicted reachable region.

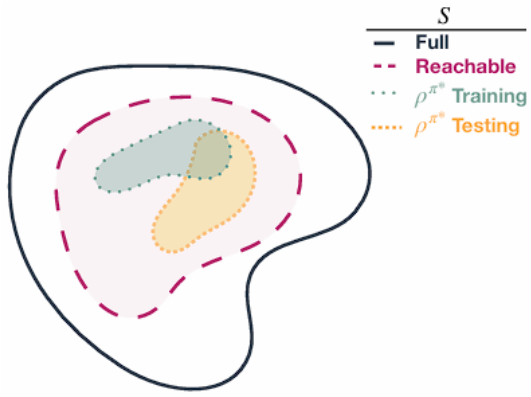


Figure 2: Visualization of a reachable space taken from [15]

The results by Weltevrede et al. [15] have highlighted that increasing the size of the training state space proved to enhance performance on a test space, which may or may not overlap with the training set within the reachable space. This performance improvement extends even to initially unreachable starting states. This is valuable information as it shows that more diverse data can help boost generalization even in largely different tasks and will be used to test the generalization capabilities when comparing the BC and CQL methods.

3 Methodology and Experimental Setup

This section starts by covering the details of the environmental setup that is used for the experiment, followed by what datasets were chosen and the reasoning behind them. The training subsection outlines the library used for the machine learning methods and how the models are trained, followed by how hyperparameter tuning is carried out. Lastly, the section concludes by explaining how the models are evaluated on specific policies.

3.1 Environment and Tasks

Both data collection policies and testing are held in a modified FourRoom environment from Minigrid [3], a python based reinforcement learning environment built on top of the Gymnasium [14] API. The environment is a 9x9 grid divided into four rooms. Each room is separated by a wall with a door to enter a nearby room. Since the length of a wall is 3, there are 81 different possible layouts due to the variable door positions. The earlier-mentioned modifications to the environment include:

- Modifying the available actions of the agent to be only left, right, and forward.
- Transforming the observation to be four 9x9 arrays. The first one is the location of the agent, the second is the direction the agent is facing, the following one is the location of the walls and the last is the position of the goal.
- An undiscounted reward of 1 for reaching the goal.

The task for the agent is to reach the goal by traversing the environment and if done successfully a reward of 1 is given no matter the amount of steps it took to get there. If the agent fails to finish the task within 100 actions, it gets a reward of 0. The value of 100 steps was chosen, as the longest tasks would not take more than 20 actions to reach the goal. To facilitate

a multi-task setting, a different layout is used with varying goal and agent positions. This environment is chosen because of its small scale and simplicity, allowing for shorter training times and hopefully clearer interpretations of results.

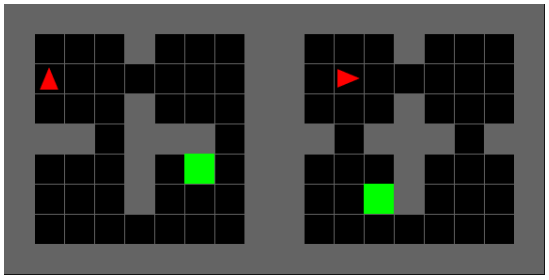


Figure 3: Minigrid visual example of 2 different tasks in the training configuration

3.2 Data Collection Policies

The policies by which the datasets are gathered consist of:

- Expert - The agent always selects optimal actions that take the shortest route to the goal. This serves as a baseline for comparing other policies.
- Expert-Suboptimal: The agent has a 50% chance to either choose the action an expert would take or an action provided by a DQN. This dataset aims to compare with the baseline how more diverse data helps the methods generalize better and if increasing the dataset size also significantly impacts performance.
- Random Walk: The agent first moves randomly for a number of steps ranging from 10 to 50. The range was chosen because determining a specific step count was challenging. The lower bound of 10 ensures the agent moves a short distance, while the upper bound of 50 is two to three times the length of the longer Expert policy paths. After the random walk, the agent follows an optimal path to the goal. Like the Expert-Suboptimal dataset, this dataset tests whether diversity improves generalization skills. Additionally, it also simulates a slight task variation by moving the agent away from its starting position and having it reach the goal with an optimal trajectory. Since the idea behind this dataset is to try to add different starting positions, the optimal trajectory from the original starting position is also added once.

The data collection agent is run one or more times for each training environment in order to gather episodes that are saved as a dataset. Each episode records the observations, actions taken, and rewards gained by the agent from start to finish, ending when the goal is reached. The only exception is the expert policy, which consistently produces the same trajectory regardless of how often the agent is run. The specific step counts chosen for each dataset are 1000, 5000, 10000, 25000, and 100000 to study the effect of dataset size on the performance of the models.

Data is gathered three times using the seeds for data collection policies that use randomization during the action selection process [1, 2, 3]. This means that while the expert policy results in a single dataset, the Expert-Suboptimal policy generates three datasets for use in training and evaluation.

3.3 Models and Training

Both BC and CQL methods are from d3rply [13], a Python-based machine learning library offering many algorithms for discrete and continuous datasets. With the current environment configurations, the discrete versions are used. These methods are trained on datasets collected by various policies mentioned in the previous subsection. The training set consists of 40 different tasks, each with a varying topology. This number of tasks was chosen in order to not overtrain the models on all of the available topology configurations, as that would prevent determining how well the models can generalize to unseen configurations.

Models are trained from 1000 steps to 50000 steps with the model being saved every 1000 steps for later evaluation. This is done because the validation set tends to be flaky and unreliable, making early stopping ineffective. 50000 steps are the cutoff point because some experimentation and manual inspection have shown that training beyond often leads to significant overfitting.

3.4 Hyperparameter Tuning

Hyperparameter optimization uses the Optuna framework [1], which efficiently searches large spaces with its default hyperparameter sampler based on the Tree-structured Parzen Estimator (TPE) algorithm. In each trial, TPE fits a Gaussian Mixture Model (GMM) $l(x)$ to the best parameter values and another GMM $g(x)$ to the rest, selecting the parameter value x that maximizes the ratio $\frac{l(x)}{g(x)}$. The optimizer is executed for each dataset associated with a policy, using seeds [5, 6, 7, 8, 9]. Details on the hyperparameters and search spaces used in the study can be found in Appendix A. To ensure a fair comparison between BC and CQL, Optuna conducts 50 trials for both models

Typically, hyperparameter optimization is performed on a validation set, but this approach is challenging in our experimental setup due to the limited number of training environments. The validation set yields inconsistent or minimal rewards, especially for the CQL method. To avoid losing valuable data on an unreliable validation set for CQL, hyperparameter optimization is performed over the entire training set. Once the parameters are set, models are trained using seeds [0, 4219, 17333, 39779, 44987], generated by a random prime generator and used for each dataset that a policy may have.

3.5 Evaluation

Methods are evaluated by plotting the average of average agent rewards in a training range of 1,000 to 50,000 steps. The performance of an agent is determined by calculating the mean reward for a specific training step count across various datasets and training seeds. The evaluation itself is conducted in two types of environments:

- Reachable Environments: These consist of 40 tasks with topologies that the agent has already encountered during training. The agent might have learned and adapted to these layouts, allowing us to assess its performance on familiar tasks.
- Unreachable Environments: These consist of 40 layouts that the agent has not seen during training. Testing in these environments helps us evaluate the agent's ability to generalize to new scenarios.

For example, a policy with 3 collected datasets and 5 training seeds would produce 15 agents trained for 1,000 steps. These 15 agents would then be evaluated in various environments specified in an environment configuration file. The average rewards for these 15 agents are further averaged to determine the overall performance of the policy after 1,000 steps of

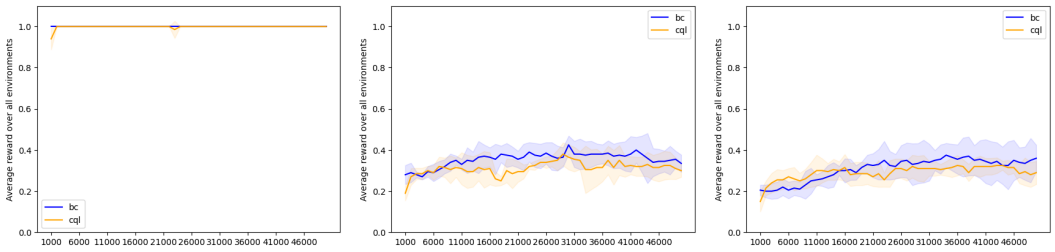
training, based on 3 datasets and 5 seeds. This process would be repeated in 1,000-step increments up to 50,000 steps and the results would be plotted for comparison.

4 Results

This section covers the results of the models trained on the singular dataset collected by the Expert policy and the datasets of size 1000, 10000, and 100000 by the Expert-Suboptimal and Random Walk policies. Datasets of size 5000 and 25000 have been placed in Appendix B as they provide minimal increases and have almost nothing interesting to show. All of the graphs listed here depict the average rewards of agents trained from 1,000 to 50,000 steps in 1,000-step increments.

4.1 Expert dataset collection policy

Both BC and CQL perform equally well when tested on the training set, shown in Figure 4a, with CQL not being able to finish once or twice just a couple of times. It is interesting to note that by 2000 training steps both methods reach a mean reward of 1, meaning that they have mastered the training set. This makes sense since both methods use batch learning and the dataset is only 372 transitions long. Looking at the Figures for reachable 4b and unreachable 4c environment sets, BC slightly outperforms CQL but still stays within the confidence interval.



(a) (b) (c)

Figure 4: Average Reward of the Expert policy on (a) the training, (b) the reachable and (c) unreachable sets

4.2 Expert-Suboptimal Dataset Collection Policy

Looking at the training column of Figure 5, the average reward mostly falls within the range of 0.75 to 0.8. BC tends to learn slightly faster than CQL, but even CQL gets to the previously mentioned range within 6000 thousand training steps. For this policy, dataset size has no substantial results, with the means increasing by a marginal amount.

In the reachable set, the dataset made up of 1,000 has mostly equal results for BC and CQL, with the imitation learning method producing slightly better results in the lower training ranges and vice versa, as depicted in Figure 5b. The same trend continues for the 10000 transition dataset results in Figure 5e, with CQL almost surpassing the confidence interval by the end. Still, neither BC nor CQL outperform each other as their results are largely within each others' confidence intervals. On 100,000 transition datasets 5h, BC generally outperforms CQL beyond the confidence interval bounds, leading by approximately 0.1 in score until the 100,000 step mark, where CQL nearly catches up. The overall increase in

performance from 1,000 to 100,000 steps is about 60% for BC and around 40% for CQL, with increases corresponding to dataset size.

For the unreachable set, both methods perform similarly when trained on the dataset of size 1000 (Figure 5c). When examining the models trained on 10000 transitions, CQL has a slight advantage outside of confidence intervals over BC. However, the lead disappears in the results of the 100,000 transition dataset (Figure 6i), where both methods again are largely equal. The overall increase from 1,000 to 100,000 steps is about 40% for BC and CQL.

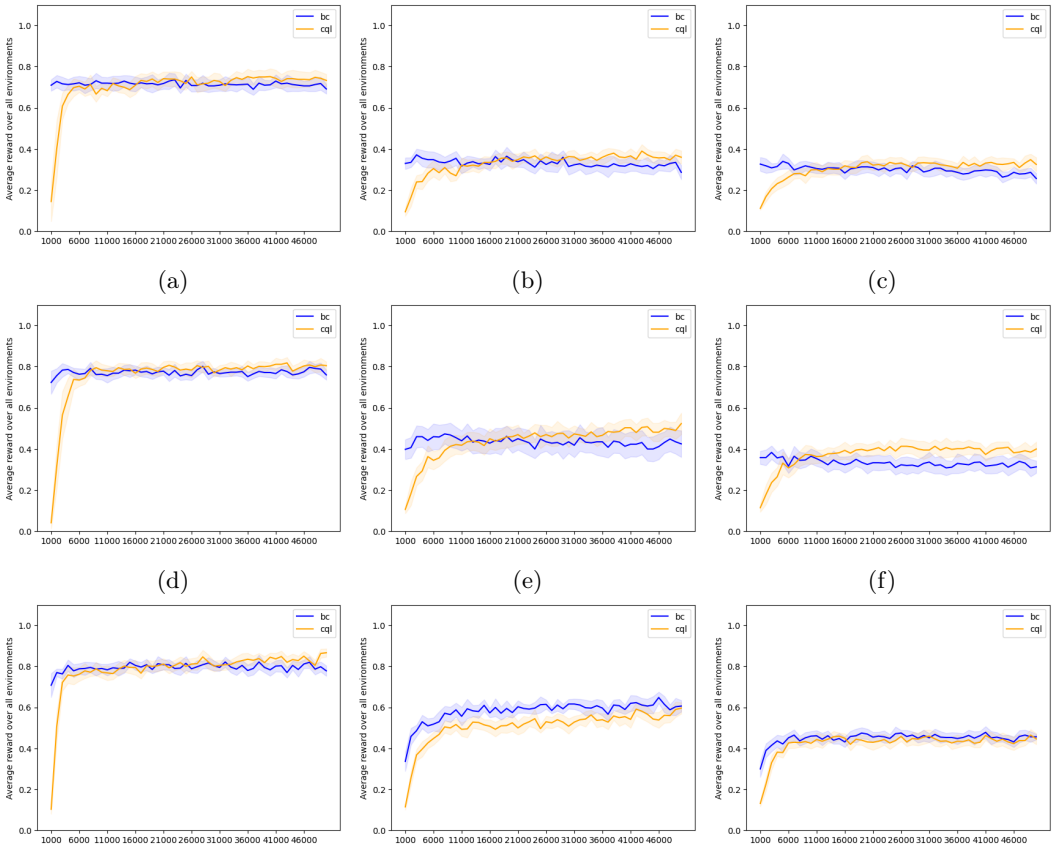


Figure 5: Average Reward of the Expert-Suboptimal Policy: The rows represent datasets with 1000, 10000, and 100,000 transitions from top to bottom. The columns show the average reward on the training, reachable, and unreachable environment sets in the specified order

4.3 Random Walk Dataset Collection Policy

Looking at the smallest dataset from Figure 6a, the training results are more or less the same. However, with larger datasets, CQL agents achieve significantly higher scores compared to BC, although the learning period is somewhat longer.

The reachable results, shown in Figure 6b, depict that BC has consistently higher rewards throughout the entire training range. This is no longer the case for the dataset of size 10000 6e, where it only scores higher at the lower training step range with the rest being equal. Looking at the models trained on the 100000 dataset, BC again performs slightly better initially but is eventually surpassed by CQL, where the Q-Learning method reaches a record high mean

reward of 0.8, as shown in Figure 6h. Comparing Figures 6b and 6h, it can be seen that the improvement for CQL is around 3 to 4 times and roughly 2 to 2.5 for BC due to the added data.

In the unreachable configuration, both methods produce similar results with CQL only exceeding the confidence interval towards the end of the training range as shown in Figure 6i, which depicts the results of the 100000 size dataset. Both methods have a similar increase of 70% for the initial part of the training range, but CQL nearly doubles its average reward when compared to the rewards of Figure 6c near the latter part, reaching almost 100%, while BC remains at roughly 70%.

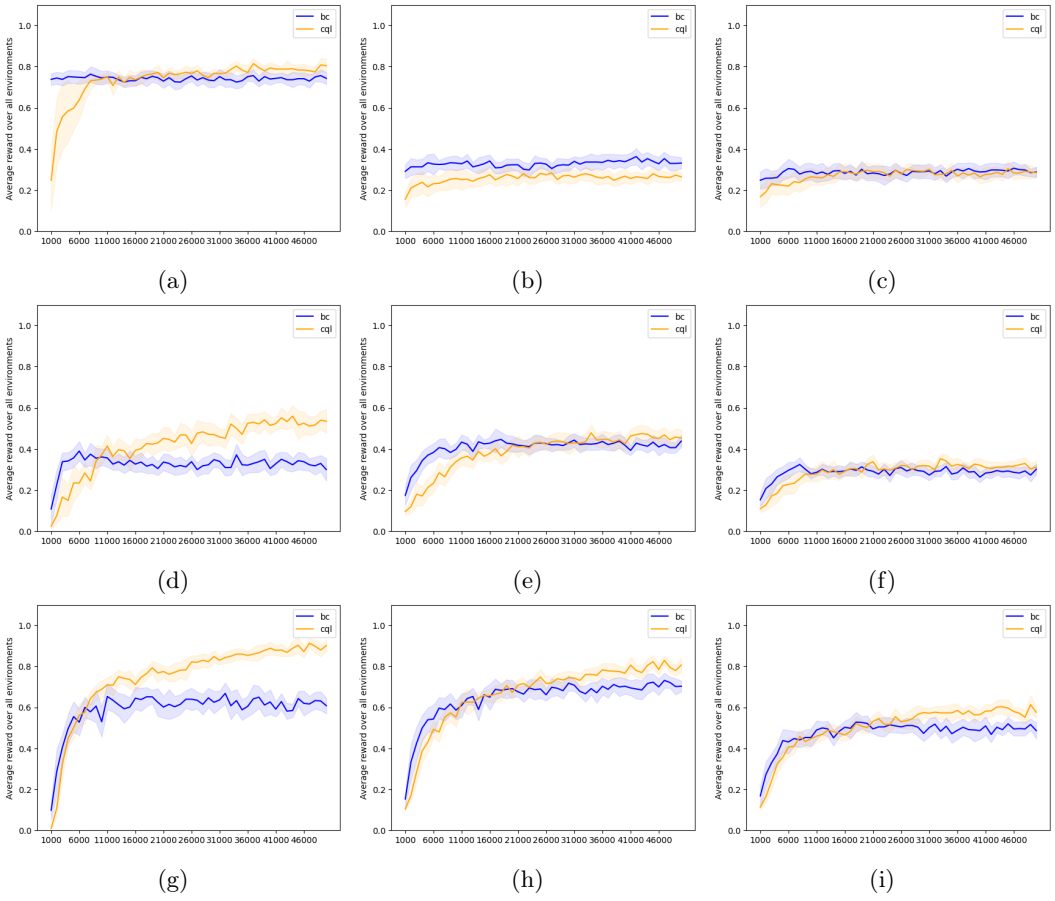


Figure 6: Average Reward of the Random Walk Policy: The rows represent datasets with 1,000, 10,000, and 100,000 transitions from top to bottom. The columns show the average reward on the training, reachable, and unreachable environment sets in the specified order

5 Responsible Research

The entire codebase relating to this experiment is shared in the Github repository https://github.com/amphities/rp_cql. This entire study tried to adhere to the FAIR principles:

- Findable - the various environment configurations, datasets and hyperparameters used in the training of the models can be found in the publicly shared repository.

- Accessible - the public repository can be easily cloned with all of the contents, besides the models as they take up over 30GB of space. There are also examples of how to access the datasets and hyperparameters.
- Interoperable - the repository has a *requirement.txt* file which can be used in combination with pip to install all of the needed libraries for running the codebase. Since everything was done on a Linux there are no guarantees that it would work for different operating systems. Moreover, there are examples of how to train the different models on various dataset and hyperparameter combinations and how to get and plot their rewards over different testing configurations.
- Reusable - dataset gathering policies and how to use them are explained in the READ_ME file or in the paper which is linked in the previously mentioned file. This should be enough for data reusability. As for hyperparameter optimization and model training, the used seeds are mentioned in Section 3.2 of this paper.

6 Discussion

This section focuses on expanding on the gathered results and highlights some limitations that may have had a big influence on the outcomes.

6.1 Policy Comparison for the Test Environments

Regarding the test set, all policies fail to match the rewards of the models trained on the Expert policy datasets. This may be attributed to possibly conflicting actions collected in a random or suboptimal manner. Notably, there is also a steep drop in test performance between the Expert-Suboptimal and Random Walk policies. This might be due to the fact that the Random Walk policy has a more randomized nature, by first having 10 to 50 random actions taken before following an optimal trajectory. In contrast, the Expert-Suboptimal policy only includes random actions half of the time with the other half being optimal and potentially leading to a more stable trajectory. This may also be the reason behind the almost identical graphs in the test set of Figure 5.

The rewards from Figure 6a are outliers compared to the lower graphs, possibly because it contains only 60 episodes, meaning the agent experienced each environment only once or twice. Due to that the training data might conflict and be easier to learn from. Another interesting observation is that BC generally reaches its highest performance faster compared to CQL, though this might be because of the larger batch sizes selected as a hyperparameter. Lastly, the big separation appearing between the methods in the Random Walk testing graphs could be attributed to CQL managing to focus on the optimal trajectory provided in the policy among the random ones, whereas BC would try to imitate all of the data provided thus resulting in poorer test performance.

6.2 Policy Comparison for the Reachable and Unreachable Environments

Both the Expert-Suboptimal and Random Walk policies outperform the Expert when trained on datasets of size 25000 (Figures 7e and 8e) and 100000 (Figures 5h and 6h) for the reachable set. While increasing the dataset size leads to a steady improvement in performance, significant rewards beyond the confidence intervals of the Expert policy are only seen after increasing the dataset size by more than 50 times. Looking at the average reward of models trained on data collected using the Expert-Suboptimal and Random Walk methods reveals no huge

differences for the dataset sizes 1000, 5000, 10000, and 25000. The only minor observation is that it takes models slightly longer to train up to a point where additional training no longer gives substantial improvements on Random Walk data. However, this is not the case for datasets of size 100,000 and, as models trained on Random Walk data peak higher than those trained on Expert-Suboptimal data. This difference can be seen in the latter part of the graph, with the highest observed values being around 0.8 for CQL and 0.7 for BC.

Moving on to the unreachable environments, the results are largely similar to those of the reachable ones. While the means for the dataset sizes of 25000 transitions (Figures 7f and 8f) are slightly higher, they still fall within the large confidence interval of the Expert policy values. That does not hold for the 100000 transition datasets (Figures 5i and 6i), as they finally surpass the confidence intervals with higher values. As with the reachable configurations, models trained on the Random Walk dataset produced the highest recorded values seen for this set of environments, with CQL outdoing BC towards the higher step counts again. Another thing worth noting is that with more diverse data the increase in performance is seen not only in the reachable but also in unreachable environments, though the jumps are smaller. This improvement falls in line with what was described in Weltevrede et al. [15].

Lastly, it is important to highlight the gaps between training and testing performances of models trained on different data collection policies. The largest one can be seen in the Expert policy which is expected as it is difficult to generalize from optimal data alone. The second biggest gap is for the Expert-Suboptimal policy, with the gap growing smaller as the dataset size grows larger. For the last data collection method increasing the amount of transitions shrinks the gap drastically, with BC sometimes achieving even better rewards than on the training set. Overall, besides the results dropping by a tiny amount between datasets of size 1000 (Figures 6c and 8c), more data helps lessen the generalization gap in this experiment both for reachable and unreachable sets. Interestingly enough these findings do not align with the ones reported in the study by Mediratta et al. [11]. Their study suggested that, in their particular experiment, increasing the amount of data from the training levels did not have a notable increase in generalization performance.

6.3 Limitations

Due to the reasons discussed in Section 3.4, a validation set was not used in this study. Because of that, hyperparameter optimization focused on minimizing step counts over the training set rather than optimizing generalization, potentially leading to suboptimal hyperparameters. Another issue stemming from the absence of a validation set was the inability to implement early stopping. As a result, the tuning process was capped and evaluated after 20,000 training steps, which may have hindered performance for agents that might have performed better with more suitable parameters for ranges past or below 20000. Even if a validation set had been used, the choice of the hyperparameters and their ranges were also of concern as it was not much more than slightly educated guesswork. Furthermore, due to limited computing power, the search space may have not been explored thoroughly enough. More specific details about that can be found in Appendix A.

Another limitation was the use of flattened observations. Instead of employing the 4x9x9 array mentioned in Section 3.1, a flattened array of size 324 was used for model training due to its compatibility with the library. Although the models were able to learn, some information may have been lost during the conversion process, as the distinction and importance between the four arrays became less clear.

Lastly, the training seeds had a huge impact on the results. While the limited number of chosen training seeds was primarily due to computational constraints, ideally the number of training seeds should have been at least doubled. This recommendation is based on observations from a peer experiment, where hyperparameter optimization included the seed as one of

the tunable parameters, revealing its vastly higher importance regarding other parameters.

7 Conclusions

This paper extends the multi-task experiments to a different environment than in Mediratta et al. [11] and shows that in this experiment both BC and CQL generalize similarly to both familiar and unseen tasks, with either one having a small lead at some point. The only major difference between the observed methods is that CQL manages to achieve better rewards when trained on data consisting of some expert trajectories and a large number of random ones. Additionally, it shows that models trained with more diverse data collection policies, one introducing suboptimal trajectories and the other moving the agent around its start location, achieve higher rewards than those trained solely on optimal data, but only after a substantial amount of data is added.

For future experiments comparing BC and CQL, it may be beneficial to use a proper validation environment to enable hyperparameter optimization aimed at improving generalization over the validation set. Additionally, test results for both methods may improve by using non-flattened observations. Another notable issue is the significant impact of seed choice on test results. The ideal number of seeds should be higher for future experiments. Lastly, observing the differing results in the training graphs of models trained on the more diverse Random Walk dataset suggests that it may be promising to conduct a similar experiment as done in Weltevrede et al. [15]. The experiment uses a decaying random action factor, due to which the data collection agent starts with gathering diverse data and then shifts to more optimal actions. This could provide a more extensive comparison of how these methods handle generalization under different levels of diverse data.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[2] S. Akshay, N. Bertrand, S. Haddad, and L. HÃ©louÃ«t. The steady-state control problem for markov decision processes. volume 8054 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, 2013. Springer.

[3] Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

[4] Ben Hambly, Renyuan Xu, and Huining Yang. Recent advances in reinforcement learning in finance. *arXiv preprint arXiv:2112.04553*, 2021. 60 pages, 1 figure.

[5] Athanasios Kapoutsis. The monster of distribution shift in offline rl and how to pacify it. `https://medium.com/@athanasios.kapoutsis/the-monster-of-distribution-shift-in-offline-rl-and-how-to-pacify-it-4ea9a5db043`, 2022.

[6] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day, 2018.

[7] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2017.

[8] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning, 2021.

[9] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning, 2020.

[10] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020.

[11] Ishita Mediratta, Qingfei You, Minqi Jiang, and Roberta Raileanu. The generalization gap in offline reinforcement learning, 2024.

[12] Ofir Nachum and Mengjiao Yang. Provable representation learning for imitation with contrastive fourier features, 2021.

[13] Takuma Seno and Michita Imai. d3rlpy: An offline deep reinforcement learning library. *Journal of Machine Learning Research*, 23(315):1–20, 2022.

[14] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel GoulÃ£o, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea PierrÃ©, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

[15] Max Weltevrede, Matthijs T. J. Spaan, and Wendelin BÃ¶hmer. The role of diverse replay for generalisation in reinforcement learning, 2023.

A Hyperparameter Tables

A.1 BC Hyperparameters

Dataset	Seed	Learning rate	Batch size	Beta
optimal	1	0.0004024066464714613	50	1.5528833190894193
mixed_suboptimal_1000	1	0.002995896368344014	50	9.581982606170591
	2	0.0016694607198667812	25	9.69100487800163
	3	0.0009050743092524423	100	8.639215346071603
mixed_suboptimal_5000	1	0.00437156938274838	25	2.611742295897904
	2	0.0015503930592067948	25	5.516085048647968
	3	0.0037892970949022463	25	1.1219374651186562
mixed_suboptimal_10000	1	0.003994996930608538	100	7.2697299341324415
	2	0.0019183473631971704	50	0.21920144469111857
	3	0.0030446962649238883	25	8.708090147341686
mixed_suboptimal_25000	1	0.0037892970949022463	25	1.1219374651186562
	2	0.0005733759460591727	100	6.168049460735248
	3	0.0020079398402026555	50	2.8917228775849653
mixed_suboptimal_100000	1	0.0002686370854445551	25	4.220902108786815
	2	0.000821695790470142	25	8.26184335569783
	3	0.00048068701671538725	50	7.004209087416205
random_walk_1000	1	0.0007161010103624337	100	4.958078673854734
	2	0.0004024066464714613	50	1.5528833190894193
	3	0.0014546618326509637	100	3.872850280170462
random_walk_5000	1	0.004881819543815398	50	1.433482936107339
	2	0.0030389864559298677	50	9.69100487800163
	3	0.0008703948385351864	100	9.72624849387303
random_walk_10000	1	0.00013733487824847007	50	0.2497625452340968
	2	0.0008776430649577643	25	3.427896690034922
	3	0.0003698289658343733	100	3.547165314029729
random_walk_25000	1	0.0014087154848103005	25	2.3665170866316867
	2	0.0004024066464714613	50	1.5528833190894193
	3	0.0010401035925835158	25	3.7842493412334477
random_walk_100000	1	0.0003856835463327651	25	1.838970857682466
	2	0.0011650124592172252	100	6.949583428109686
	3	0.0005057395077277829	50	3.547165314029729

Table 1: BC hyperparameters. The optimal dataset is gathered by the Expert, mixed_suboptimal by the Expert-Suboptimal and random_walk by the Random Walk policies. The number next to the dataset is how many transitions they contain.

The reasons for choosing the learning rate and batch size as hyperparameters were that it is commonly used in most model tunings. The range for the learning rate was from 0.01 to 0.0001 as those were the values I've seen in other papers while researching the topic. That is not the case for batch size and beta though as it was guesswork for what values to use. In the d3rlpy [13] library the discrete BC implementation has a default batch size of 100 and some studies [7] have shown that smaller batch sizes can provide better generalization results so with that in mind the options of 25, 50, 100 were chosen to be picked for sampling. The last parameter is beta, a regularization factor that may help with limiting model complexity and improving generalization. The value range for this parameter was chosen to be from 0.1

to 10 with no other reason than the default in the library being 1.

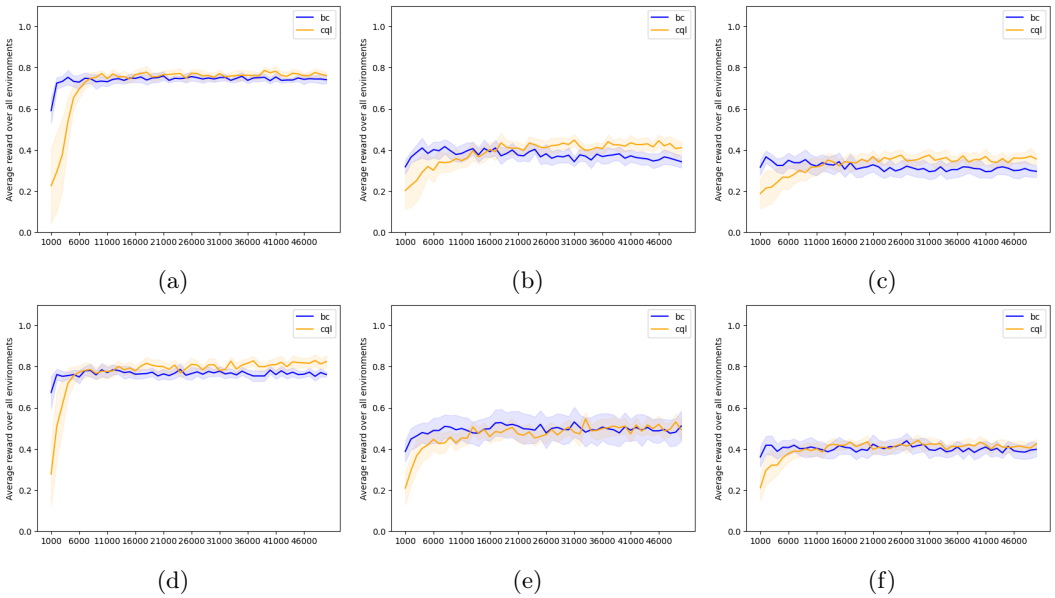
A.2 CQL Hyperparameters

Dataset	Seed	Learning rate	Batch size	Alpha
optimal	1	0.0004024066464714613	16	0.8191038650038539
mixed_suboptimal_1000	1	0.0001688283764578491	32	3.139042958541156
	2	0.00018213346106928585	8	1.8180478568320502
	3	0.00010487087970761692	32	1.8518903714390942
mixed_suboptimal_5000	1	0.00012567457510646065	8	4.026024184214208
	2	0.00015299139796093366	8	1.2440298176140852
	3	0.0018186511139638279	32	1.196994133507301
mixed_suboptimal_10000	1	0.00010392030527469727	32	1.8946542481974347
	2	0.000804590469607637	8	2.647514826963019
	3	0.00015608194420524345	8	0.5918069439821343
mixed_suboptimal_25000	1	0.002731097500696576	32	0.6877066139086498
	2	0.00010701359820279658	16	1.0871021215726497
	3	0.00037631426125345945	16	0.7371400033795606
mixed_suboptimal_100000	1	0.000154583397223047	32	1.5056463447124997
	2	0.0002342012033810673	16	0.6041559387762274
	3	0.0004975415182415506	16	0.1300732583169888
random_walk_1000	1	0.0029049221149668814	32	3.429750853114194
	2	0.0001510490053966706	16	0.12319366564278567
	3	0.0002241282807136287	32	3.671861120052074
random_walk_5000	1	0.00010007322954979191	8	0.9879469275223968
	2	0.00010455540315546114	8	1.7512992645747365
	3	0.0005049410283573611	32	3.353302217646795
random_walk_10000	1	0.0006107246972142488	32	0.44745772548764423
	2	0.00021659235092409123	16	0.3144042490238427
	3	0.0002961704271358361	8	0.46554429139135994
random_walk_25000	1	0.0007599552280003335	32	0.3748740896436084
	2	0.0009658679787269233	32	0.42792875914747386
	3	0.0001329490254619179	32	0.49306053459178867
random_walk_100000	1	0.00028339409073761037	32	1.166516663871695
	2	0.0005067132513080573	32	1.5391038201025486
	3	0.00030899394959249716	32	0.8786269465962936

Table 2: CQL hyperparameters. The optimal dataset is gathered by the Expert, mixed_suboptimal by the Expert-Suboptimal and random_walk by the Random Walk policies. The number next to the dataset is how many transitions they contain.

The reasons behind choosing learning rate and batch size remain the same as in CQL with also the same learning rate being used for CQL. The batch sizes used for sampling were [8, 16, 32] with the reasoning being the same as in the BC section just that this time the default value for CQL was 32 in the d3rlpy [13] library. The last parameter called alpha is used to regulate how conservative is the Q-Value update function 4 is a higher value meaning more importance and 0 meaning that it is just a regular Bellman update.

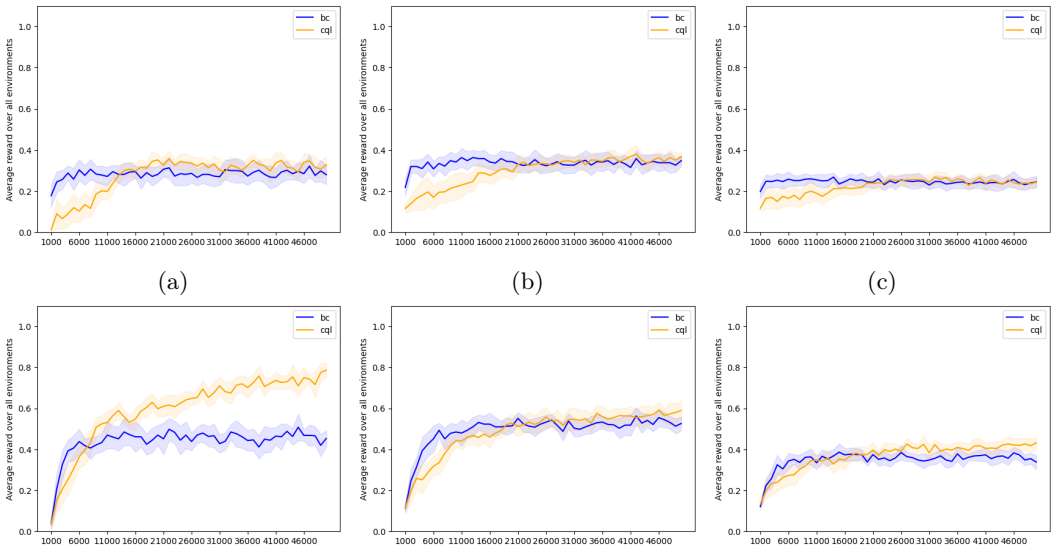
B Results for datasets of size 5000 and 25000 for Expert-Suboptimal and Random Walk policies



(a) (b) (c)

(d) (e) (f)

Figure 7: Average Reward of the Expert-Suboptimal Policy: The rows represent datasets with 5,000 and 25,000 transitions from top to bottom. The columns show the average reward on the training, reachable, and unreachable environment sets in the specified order



(a) (b) (c)

(d) (e) (f)

Figure 8: Average Reward of the Random Walk Policy: The rows represent datasets with 5,000 and 25,000 transitions from top to bottom. The columns show the average reward on the training, reachable, and unreachable environment sets in the specified order