Generalization in Offline Reinforcement Learning: Comparing Implicit Q-Learning with Behavioral Cloning

Juan Jose Tarazona Rodriguez[1]
Supervisors: Dr. Matthijs Spaan[1]**, Max Weltevrede**[1]
[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: Juan Jose Tarazona Rodriguez
Final project course: CSE3000 Research Project
Thesis committee: Dr. Matthijs Spaan, Max Weltevrede, Dr. Elena Congeduti

Abstract

Offline Reinforcement Learning (Offline RL) involves learning policies from a static dataset without further interactions with the environment, making it suitable for high-stakes scenarios where data collection is costly or risky. This paper investigates the generalization capabilities of Implicit Q-Learning (IQL), an offline RL algorithm, compared to Behavioral Cloning (BC). We adapt the IQL algorithm for discrete control and evaluate both IQL and BC in a four-room environment using training datasets generated from different behavioral policies. Performance is assessed based on average rewards over various test seeds, on reachable and unreachable tasks, as well as the training set. Our results indicate that BC consistently outperforms IQL across all scenarios, although IQL reaches peak performance faster. This study highlights the need for further research into offline RL algorithms for better generalization and more robust performance in diverse environments. Full code available on GitHub.

1 Introduction

Reinforcement Learning (RL) involves an agent learning optimal behaviors by interacting with its environment via trial and error [1]. RL has been successfully applied to a wide variety of tasks, including the fields of healthcare, robotics, transportation and finance [2]. However, RL remains an active research topic due to the many challenges it faces. In particular, we are interested in the safety problems that might arise when collecting data through real-world world exploration [3]. This is especially impactful in areas where the agent has to interact with humans.

This leads to the development of offline reinforcement learning. The main difference with traditional reinforcement learning is that in offline RL [4], the agent is unable to interact with the environment to collect additional transition data using its behavior policy. Instead, the algorithm learns the best policy it can from a static dataset. This approach is particularly useful in scenarios where data collection can be expensive or dangerous.

Recently, research in the field of offline reinforcement learning has encountered an important problem, as argued in [5], which is that the generalization of offline RL methods to new environments remains largely unexplored. Particularly, it shows that offline RL fails to outperform behavioral cloning (BC) methods in solving similar but new tasks (multi-task setting). This means that offline RL does not find a better policy than the behavior policy. The paper accurately argues that it is essential to solve this performance issue, given that the applications of offline RL are on high-stakes scenarios where safety is the utmost priority. However, [5] focuses specifically on levels from the ProcGen [6] benchmark, which may be limiting in terms of assessing generalization capabilities definitively. Moreover, it only considers training sets generated by expert data as well as mixed expert-suboptimal data, defined to be giving approximately 75% of the returns of the expert data. It does not use other types of training sets generated by, for instance, random actions, which could provide different insights to the peformance question.

Bias can be a significant issue in machine learning, particularly reinforcement learning research [7] and given the recency of these results, it is important to evaluate them more extensively. Additionally, as illustrated in [8], generalization of offline RL methods can vary over datasets, with more extensive ones improving this aspect. Therefore, exploring a dif-

ferent environment and differently structured datasets is important.

Among recent offline RL algorithms, Implicit Q-Learning [9] has shown outstanding performance over various benchmark tasks from D4RL [9, 10] and demonstrates effectiveness across other datasets as well [8]. As this algorithm is considered a state-of-the-art method for offline RL and is evaluated in [5], we will examine its performance in the multi-task setting. Other works have suggested using diffusion models to explore generalization in the multi-task offline RL setting [11], which are improved upon in an alternative interpretation of IQL [12]. However, these methods are not yet fully consolidated, therefore we will keep our focus on the classic IQL method. As such, we aim to answer the following research question:

To what extent does Implicit Q-Learning enable generalization, and how is this capacity influenced by the dataset composition?

The main contributions of this paper are:

- To investigate the reproducibility of the results presented in [5] for IQL compared to BC, which we find to align with our results.
- To explore the influence of different compositions of the training sets on the generalization capabilities of IQL.
- To show the possibility, given enough data, to learn from completely random actions.

The rest of the paper will be organized as follows: in chapter 2, the background required to understand this research is covered. Chapter 3 presents Implicit Q-Learning (IQL) as well as the benchmark comparison algorithm, Behavioral Cloning (BC). Chapter 4 then follows with a description of the environment and experimental setup used, and chapter 5 discusses the results, answering the generalization performance question. In chapter 6 we summarize our findings and future work possibilities, concluding the paper. Chapter 7 covers responsible research practices used throughout our work.

2 Preliminaries

In this section we cover reinforcement learning and offline reinforcement learning definitions to ensure understanding of the algorithms whose performance we are interested in. Moreover, the methodology used to evaluate the generalization capability of IQL is briefly introduced.

2.1 Reinforcement Learning

The reinforcement learning problem can be defined as a Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, r, \gamma)$ where $\mathcal{S}$ is a set of possible states, $\mathcal{A}$ is a set of possible actions, $T(s_{t+1}|s_t, a_t)$ is a conditional probability that defines the dynamics of the system, $d_0(s_0)$ defines the initial state distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and γ is the discount factor [4]. The goal of reinforcement learning is to learn a policy $\pi(a_t|s_t)$, that is to say, a way to determine the next action based on the current state. The optimal policy π^* to learn maximizes cumulative discounted returns [9]:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 \sim d_0(\cdot), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim T(\cdot|s_t, a_t)\right] \tag{1}$$

where the initial state is drawn from d_0, the actions are drawn from the policy and the following states are drawn from the environment dynamics.

2.2 Offline Reinforcement Learning

The algorithm in offline RL cannot interact with the environment to collect additional information on transitions and is instead provided with a static dataset from which it has to learn the optimal policy: $\mathcal{D} = \{(s_t^i, a_t^i, s_{t+1}^i, r_t^i)\}$ [4]. In essence, the dataset contains entries with a state, the action taken, the state it leads to and the reward obtained from this sequence. In some occasions, each entry also contains a flag indicating whether we have reached a terminal state.

The data for this dataset $\mathcal{D}$ has been collected using some behavioral policy π_β. The aim of offline reinforcement learning algorithms is to achieve a new policy π_{off} that outperforms at least the policy used for data collection [8]. The objective of offline RL remains the same as that from online RL, namely to find the optimal policy.

2.3 Evaluation by benchmarking

As is often the case, this study will use BC as a baseline [8]. BC is a method attempting to mimic the behavioral policy used to collect the data in $\mathcal{D}$. The first purpose of offline RL algorithms is to outperform the collection policy, therefore they should be able to outperform behavioral cloning. Moreover, we are interested in testing the conclusions from [5] on different datasets. Therefore, the method used to evaluate the generalization of IQL will be an experimental study organized as follows. First, training sets will be generated using various policies. Then, both BC and IQL will be trained on the same sets and evaluated on the same test sets. Finally, a direct comparison will be established in average rewards obtained by the algorithms.

3 Algorithms

In this section we provide a technical explanation of both Implicit Q-Learning (IQL) and Behavioral Cloning (BC) to expand the understanding of the algorithms we are comparing.

3.1 Behavioral Cloning

BC is an algorithm first introduced in ALVINN (Autonomous Land Vehicle in Neural Network) [13] and is a part of a larger category of algorithms know as imitation learning. The basic idea of BC is to cast the imitation of the behavioral policy used to obtain the dataset into a supervised learning problem, yielding promising results in terms of generalization capabilities, despite major flaws in the original algorithm [14].

The supervised learning problem aims to minimize the discrepancy between the behavior policy and the actions taken by the learned policy. This can be formalized using a loss function:

$$L(\theta) = \frac{1}{N}\sum_{t=1}^{N} \ell(\pi_\theta(a|s_t), \pi_\beta(a|s_t)) \tag{2}$$

where π_θ is the learned policy, π_β is the behavior policy, ℓ is the loss function and N the number of samples. Commonly used loss functions include the mean squared error or cross-entropy loss. We will be making use of the discrete BC implementation given by the *d3rlpy* offline RL library [15]. In this library, the loss function used is computed by cross-entropy with an added regularization term:

$$L(\theta) = \mathbb{E}_{a_t, s_t \sim \mathcal{D}}\Big[-\sum_a \pi_\beta(a|s_t) \log \pi_\theta(a|s_t)\Big] + c \cdot \text{penalty} \tag{3}$$

where $\mathcal{D}$ is the dataset to learn from, c is a control parameter for the regularization term and the penalty is the mean of the squared logits of a categorical distribution. After computation of the loss function the learned policy is updated. This is repeated until some termination criteria.

3.2 Implicit Q-Learning

IQL is a recent offline RL algorithm introduced in [9]. It suggests a novel method to combat distributional shift (incorrect estimations for actions and states outside the learning dataset). In short, the algorithm aims to estimate the maximum Q-value over actions that are in support of the data distribution without querying it on out-of-sample actions [9]. Particularly, the value function they aim to learn can be given by:

$$L(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\big[(r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q_{\hat{\theta}}(s', a') - Q_\theta(s,a))^2\big] \tag{4}$$

subject to $\pi_\beta(a'|s') > 0$. $\mathcal{D}$ is again the dataset, $Q_{\hat{\theta}}$ is a target network and Q_θ is the parameterized Q-function. The authors propose to avoid querying out-of-sample actions by using expectile regression. More specifically, an upper expectile is calculated to approximate the maximum of $r(s,a) + \gamma Q_{\hat{\theta}}(s', a')$ over actions only from the dataset. However, this leads to stochasticity due to environment dynamics. Therefore, an additional value function $V_\psi(s)$ is proposed to estimate an expectile only using the action distribution. As such, the Q-function can be estimated using an MSE (Mean Squared Error) loss, avoiding stochasticity. The value function and Q-function losses are given by:

$$\begin{cases} L_V(\psi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\big[L_2^\tau(Q_{\hat{\theta}}(s,a) - V_\psi(s))\big] \\ L_Q(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\big[(r(s,a) + \gamma V_\psi(s') - Q_\theta(s,a))^2\big] \end{cases} \tag{5}$$

where L_2^τ is an asymmetric ℓ_2 loss. With this model, we can learn an optimal Q-function but this does not entirely specify the policy that will be learned. Therefore, to conclude, the authors present a policy extracion method using advantage-weighted regression:

$$L_\pi(\phi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\big[\exp(\beta(Q_{\hat{\theta}}(s,a) - V_\psi(s)))\log \pi_\phi(a|s)\big] \tag{6}$$

where β is the inverse temperature, the lower the more similar to BC, and the higher the more it attempts to extract the optimal Q-function. Overall, the algorithm first fits V_ψ and the Q-function, and then extracts the policy using stochastic gradient descent on equation 6.

IQL offers the benefits of simplicity and efficiency, while providing state-of-the-art performance on common benchmark datasets for offline RL algorithms. We choose to use the CORL [16] library for the implementation of IQL, adapted to discrete control for our environment. The adaptations are given in appendix A.

4 Evaluation

As previously discussed, we will be comparing BC with IQL by means of running both algorithms on the same training and test sets. We will test whether BC outperforms IQL in terms of average reward obtained, thereby supporting the hypothesis that the generalization capabilities of IQL are limited. We will also test if the composition of the training sets used affects said generalization capability, and if it is possible to learn from random data. This section covers the environment used, the experimental setup and the results obtained from running the algorithms. An overview of the process is given in figure 1.

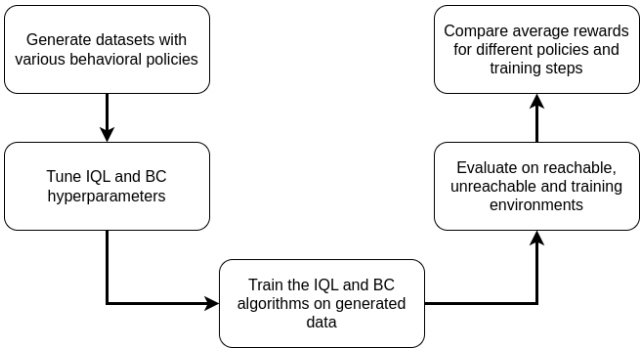
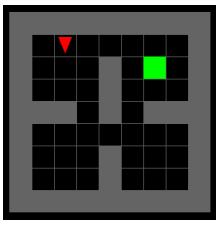


Figure 1: Flowchart overview of the methodology and experiments conducted to evaluate generalization of IQL.

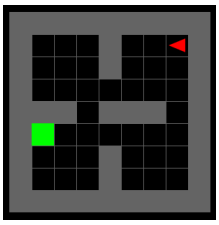
4.1 Four-room environment

For evaluation, we will use the four-room environment as presented in [17] and shown in figure 2. It has been created using the Minigrid library by Farama [18] and adapted from its FourRooms default environment. The observation, action and mission spaces are described in appendix B. We make an important distinction between reachable and unreachable generalization. A reachable state s_t is a state that can be reached by some sequence of actions performed by the agent, starting from a task seen during training. If a state cannot be reached by the agent starting from a task in the training set, it is unreachable. Figure 2 shows two states, unreachable from each other.

We will evaluate the trained models on reachable and unreachable test sets. This distinction is essential, because of how stochastic data collection policies work. A behavioral policy with some degree of random actions will add more data to a training set, but will only do so within the reachable domain, as it is the agent that takes random actions, thus not affecting the topology. Therefore, we expect to see a better performance for the algorithms when tested on the reachable set for stochastic behavioral policies.



(a) Configuration 1



(b) Configuration 2

Figure 2: Two states from the four-room environment. The agent (red arrow) has the task of reaching the goal (green square). Configuration 2 is not reachable from configuration 1 because of a different topology (wall positions) and goal location.

4.2 Dataset Generation

As explained in the introduction to this work, we want to both reproduce the results from [5] but also investigate if other dataset compositions influence the generalization capabilities of IQL. To this aim, we propose the following policies to collect actions for the datasets:

- **Expert Policy**. This policy evaluates the optimal action to take in every state. It calculates the shortest path to the goal and determines the next action as the next one of that path. Particularly, the shortest path is evaluated by transforming an observation into a graph, and using Dijkstra's algorithm.
- **Mixed Expert-Suboptimal Policy**. Similarly to [5], we also train our algorithms on a mixed expert-suboptimal policy. This policy is defined to uniformly sample between expert actions, and suboptimal actions predicted by an early stopped DQN agent approximately at 50% of the rewards of the expert.
- **Mixed Expert-Random Policy**. This policy is similar to the mixed expert-suboptimal policy but instead of a half-trained DQN agent it simply uses random actions.
- **Epsilon-greedy Policy**. This policy is similar to the mixed policies but adds the epsilon parameter. This parameter enables us to control the probability with which expert or suboptimal actions are chosen. An expert action is selected with probability $1 - \epsilon$ and a suboptimal action with probability ϵ. For the rest of this paper and evaluation, unless otherwise stated, we will use $\epsilon = 0.25$
- **Boltzmann Softmax Policy**. We propose a dynamic alternative to ϵ-greedy, namely using a Boltzmann distribution on the Q-values of the actions to determine the probability of choosing each. Particularly, the probability of an action [19]:

$$\pi_{boltz}(a|s) = \frac{e^{\hat{Q}(s,a)/\tau}}{\sum_a e^{\hat{Q}(s,a)/\tau}} \tag{7}$$

  where τ is the temperature of the system. The higher the temperature the more random the choices for actions is.
- **Fully Random Policy**. Finally, we consider a fully random policy. At each state, the action taken is randomly sampled from the action space. We use this policy to see if any type of diversity helps with generalization.

For the stochastic policies (all but the expert), three different datasets are generated with the random seeds 50, 100 and 150. The purpose of this is to get more accurate estimates in the results section hereafter, by averaging over the results obtained for each dataset. Each seed will cause the random events to occur at different times, resulting in different transitions, thus different learning.

4.3 Results

We will present and describe the results of multiple experiments to test the hypotheses from the start of this section. These results will be discussed and compared with previous work in section 5. To collect the data, both algorithms were trained on datasets collected with different behavioral policies (3 datasets per stochastic policy) for 50000 steps, with checkpoints at intermediate steps. At each checkpoint, 10 solutions were found on both reachable and unreachable test sets, repeated using 5 different random prime number seeds. The 10 solutions for each seed were averaged, and the standard deviation was calculated. This resulted in final entries differing by the algorithm used, the policy for collecting the dataset, and the training steps. The full results are in appendix D.

Hyper-parameter tuning was performed offline using an Optuna [20] study. We tuned all learning rate parameters and the inverse temperature for IQL, while network architectures remained as defaults from the libraries. Details of the implementation are given in appendix A, and tuning details are in appendix C. The best values from this tuning were used in the experiments. Notably, the actor learning rate for IQL was the only parameter that significantly impacted performance. This is accentuated when including the seed as hyper-parameter during tuning. Their importance for IQL is plotted in figure 3.

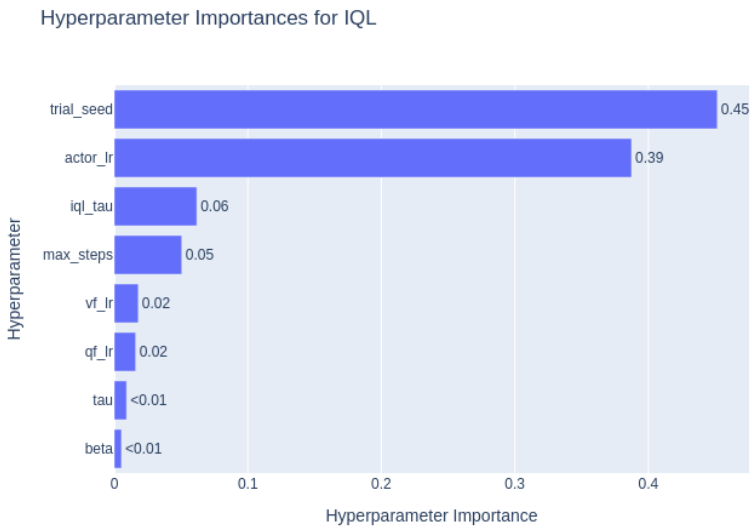


Figure 3: Relative hyper-parameter importance during tuning for IQL.

The seed showed the highest importance, indicating instability and dependence on randomness. For IQL, its importance was 0.45, and for BC, it was even higher at 0.89 as shown in appendix C. To address this while considering resource constraints, we averaged results over 5 random prime number seeds. An in-depth analysis of training steps as a hyper-parameter is discussed in subsection 4.3.4, showing they are less important but still more impactful than many learning rate parameters.

4.3.1 Testing on the training environment

Before examining generalization performance, we first investigate how the algorithms perform when trained and tested on the same environment. For this, we compare both IQL and BC at the end of the training process, namely at 50000 steps. The results are shown in figure 4.

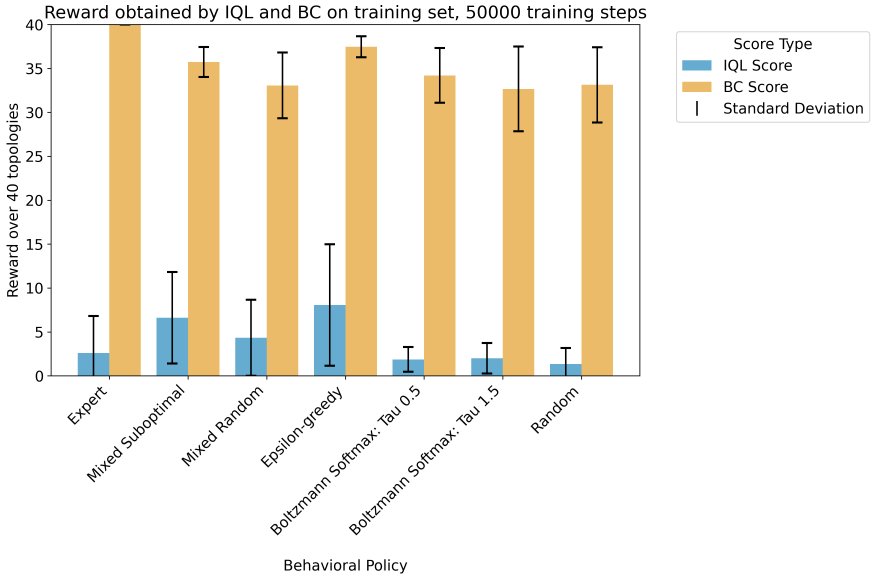


Figure 4: Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 training topologies and over 5 different seeds, trained for 50000 steps on different dataset collection policies.

We observe that IQL is severely outperformed by BC no matter what dataset collection policy was used. Contrary to expectations, IQL achieves low average rewards even on the training set, which could be caused by the environment. We can see that when trained on the expert policy, since the levels are the same and the policy was cloned, BC achieves the maximum reward. It also achieves a high average reward even when trained on other policies. However, IQL seems to benefit most from mixed datasets, especially ϵ-greedy, indicating that it could be helped by more diversity. The effect of particular dataset collection policies is looked at more in-depth in the following sections 4.3.2 and 4.3.3.

4.3.2 Effect of behavioral policies

We now conduct an experiment to compare how the algorithms perform depending on the policy used to collect the offline dataset. Particularly, we limit ourselves to the entries for 50000 steps, at the end of the training process as before, and for the reachable testing set. The results are presented in figure 5a.

For the generalization scenario, BC sees a significant drop in average reward, to around 40% of the one obtained when tested on the training set. IQL also has a drop in reward, but much less significant and only for the mixed policies, to about 80% of the training set.

We observe that Behavioral Cloning consistently outperforms Implicit Q-Learning on all datasets, achieving 160%-280% higher average rewards. For BC, expert-trained instances perform slightly better than mixed instances in terms of mean reward, while IQL benefits more from diverse data. Expert-trained instances show fewer outliers, indicating lower randomness dependence in decision-making. The mixed random policy performs slightly worse than the mixed suboptimal and ϵ-greedy policies, suggesting that not all diversity types benefit generalization to the same extent.

The fully random policy achieves lower results than other policies for both algorithms. The Boltzmann softmax policies also achieve lower average rewards, but this improves for $\tau = 0.5$, due to a stronger bias towards expert actions. Finally, the ϵ-greedy policy was run with $\epsilon = 0.25$ and shows the best performance. It can be thus deduced that there is a balance between expert actions and the benefit of including more diverse actions. Including a small amount of suboptimal or random actions, without reaching a uniform mix, appears most effective.

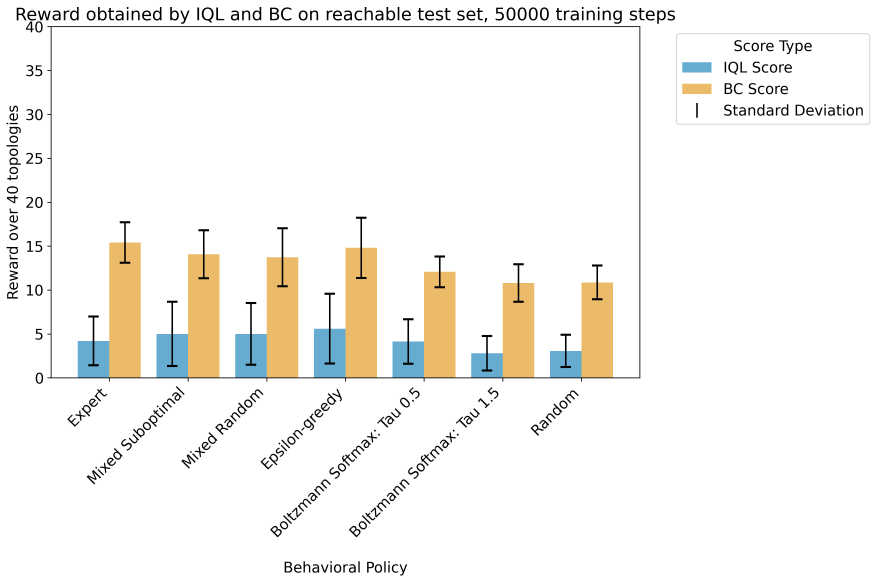
4.3.3 Reachable vs Unreachable Generalization

We run the previous experiment, but for the unreachable testing set instead. Both algorithms are still trained on all policies for 50000 steps. The results can be seen compared to the reachable test set in figure 5b.

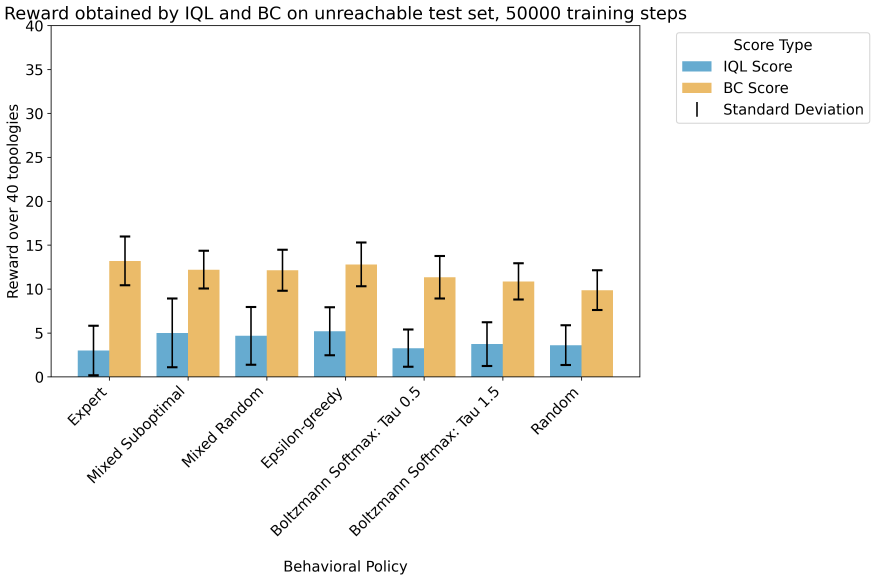
We notice, for BC, a drop in average reward of around 10% to 20% for all policies. These results are to be expected, as explained in section 4.1, given that no data is added for unreachable topologies. IQL also sees a drop in performance, except for the fully random policy, and the Boltzmann policy biased towards random actions ($\tau = 1.5$). It sees an increase in average reward, but achieves less consistency with larger standard deviations. This is also expected, since picking completely random actions should help more in unseen topologies than in seen topologies.

4.3.4 Effect of the number of training steps

We shift our focus to conduct a more in-depth study of an important hyper-parameter: the number of training steps. For this purpose, we train and test the algorithms as described at the beginning of the section. We train for 100, 200, 500, 1000, 2000, 5000, 10000, 25000, and 50000 steps, and test on both reachable and unreachable sets. The results for the reachable test set are plotted in figures 6 and 7 for IQL and BC, respectively. We examine the ϵ-greedy policy, the best out of the mixed policies. Other mixed policies follow the same trend. We



(a) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 reachable testing topologies and over 5 different seeds, trained for 50000 steps on different dataset collection policies.



(b) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 unreachable testing topologies and over 5 different seeds, trained for 50000 steps on different dataset collection policies.

Figure 5: The average rewards obtained for reachable and unreachable testing topologies.

also examine the expert and fully random policies as the two extremes.

We notice that for both algorithms, training any further than the 10000 step range results in a drop of average reward for the majority of the policies with respect to their best values, likely due to overfitting. The rewards then proceed to mostly flatline. Figure 6 shows that IQL reaches its best results when trained for approximately 500 steps, accentuated by the peak at the start. On the other hand, we can see on figure 7 that the best amount of steps for BC seems to be around 5000, with the exception of the expert policy.

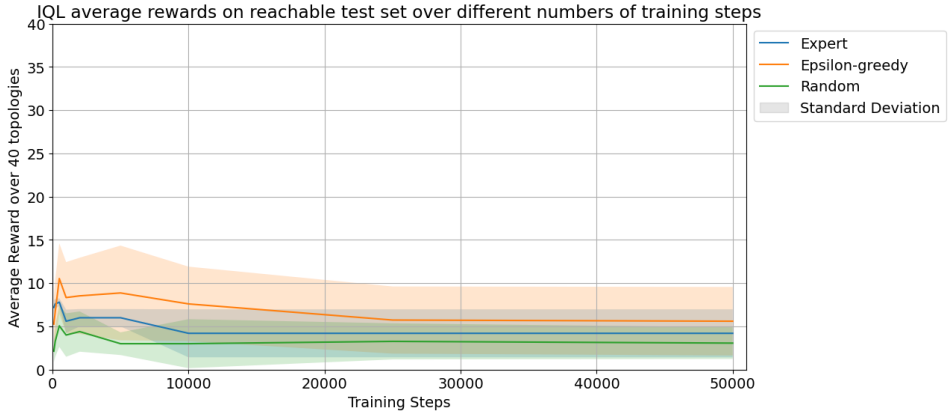


Figure 6: Average rewards obtained by IQL over different numbers of training steps for the different behavioral policies. Results averaged over 5 different seeds on the reachable topologies.

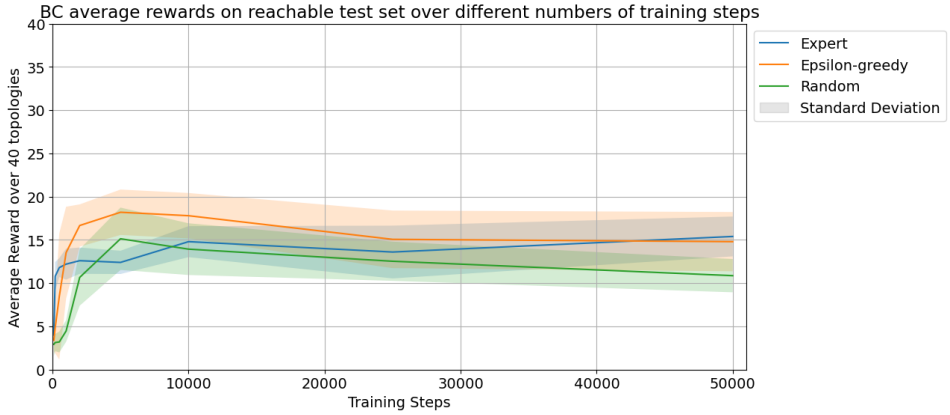


Figure 7: Average rewards obtained by BC over different numbers of training steps for the different behavioral policies. Results averaged over 5 different seeds on the reachable topologies.

Furthermore, when trained for only 100 steps, Implicit Q-Learning (IQL) outperforms Behavioral Cloning (BC), achieving average rewards between 5 and 7 compared to BC's 2 to

5 (out of 40 topologies). However, at 1000 steps, BC surpasses IQL, showing a faster improvement rate. BC reaches its peak average reward slightly slower than IQL but achieves significantly higher peaks, making it better for generalization given sufficient training steps. Similar results are observed for unreachable generalization, though BC's peaks occur even later for some policies, at 10000 steps (see appendix D). This suggests unreachable generalization requires more extensive environment learning.

5 Discussion

In the evaluation, we have compared the generalization capabilities of IQL and BC with respect to the dataset characteristics they were trained on, and the environments they were tested on. We also examined how different numbers of training steps can impact the generalization performance.

We have found that BC outperforms IQL for all dataset collection policies when tested on reachable tasks as well as exact tasks seen during training, which is in line with the results presented in [5]. Moreover, we have found that although IQL maintains its performance better than BC does, for the unreachable topologies, it is still outperformed. To expand on the results from [5], we can thus deduce that neither the environment nor the reachability affects which algorithm performs better. The dataset composition also does not seem to alter this conclusion. Additionally, we have shown that learning from fully random datasets is possible, although to a lesser extent than from the other policies for the given data.

The reason why IQL is outperformed by BC is not exactly known. It is speculated in [5] that this is not a surprising outcome, because offline RL algorithms have so far been trained and tested on the same environment. These algorithms are designed to combat the distributional shift arising from out-of-dataset actions, and IQL particularly avoids querying them altogether. Without querying outside the dataset, the method is less likely to effectively learn different tasks.

In terms of the number of training steps, the evaluation reveals that IQL does reach its peak rewards earlier than BC, but they are significantly lower. Therefore, unless in a very specific scenario where the amount of training is severely limited, BC outperforms IQL. Across experiments, we note that the standard deviation tends to be larger for mixed policies than for the expert policy. This is valuable information, as it indicates that training from an expert policy is more stable, but offers less potential in terms of average reward.

Furthermore, hyper-parameter tuning provided interesting insights. The seed is the most significant parameter, which indicates a high level of instability, possibly due to a lack of data. This limits the reliability of the results. The environment used is rather simple with a limited amount of topologies, making it difficult to strike a balance between the topologies used for training and testing. If given more training data, the algorithms may perform better for generalization, but risk overfitting. We recognise the absence of more complex environments such as MuJoCo [21], or other components of the D4RL [10] benchmark, commonly used in RL literature. These environments could provide a setting more similar to the original evaluation of IQL [9]. However, they suppose significantly larger computational costs which were not possible to incur due to hardware constraints. Moreover, the four-room environment we use enables us to easily make the distinction between reachable and

unreachable generalization, and compare their performance.

Finally, we also acknowledge that the tuning of hyper-parameters was limited to certain ranges per parameter, and amount of seeds, due to computational constraints. Target network architecture variations, such as integrating convolutional neural networks, have also not been considered. As such, we cannot discard that the outcomes could have been different while using different parameters and algorithm architectures.

6 Conclusions and Future Work

Implicit Q-Learning is a novel offline reinforcement learning algorithm offering outstanding performance on singleton environments from commonly used benchmarks, by avoiding to query out-of-sample actions. Nevertheless, it has difficulties to develop an optimal policy when dealing with similar but new tasks. This paper has evaluated the generalization performance of IQL in comparison to BC, and the impact of various dataset collection policies by answering the research question: *To what extent does Implicit Q-Learning enable generalization, and how is this capacity influenced by the dataset composition?*

We have adapted IQL for discrete control and we have examined how a different environment, behavioral policies, and hyper-parameters affect its generalization performance. In conclusion, the results are in alignment with prior work, and we find that BC outperforms IQL when trained on any policy, be it on reachable, unreachable or previously seen topologies. Additionally, we have identified the potential to learn from random data, although to a lesser extent than from other policies. The only advantage IQL seemingly has over BC is that it reaches its peak average reward faster, therefore making it at least viable in scenarios where the amount of training is severely limited. However, in such cases the average reward is not yet enough to justify the use of either algorithm.

In summary, our research indicates that IQL severly lacks in generalization performance, and, although helped by mixed collection policies, fails to yield satisfactory results. As such, we hope that our work motivates future investigation into the field of offline RL, particularly the development of algorithms that perform well in the multi-task setting.

The current implementation of IQL uses Multi-Layer Perceptrons for the target network, which require flattening observations before being able to learn from them. In future work, a promising direction could be to incorporate Convolutional Neural Netorks, which do not necessarily require this flattening, in hopes for the algorithm to learn more useful information from the observations. The environments and datasets used could also be expanded to investigate whether a different or more complex type of task can remedy the instability we have seen in the results.

7 Responsible Research

The most important ethical considerations we make are related to the potential uses of offline RL methods, the reproducibility of the research, and potential biases that could have arisen during our work.

Uses of offline RL. As indicated in the introduction, offline RL methods are generally used when training an agent online through direct interaction with the environment is either dangerous or expensive. The main concern with this lies in environments where humans can potentially be harmed, such as healthcare. Particularly, although offline RL does not directly interact with the environment during training, it will have to do so eventually when applied. Considering the findings of this research, we hope that offline RL methods will be improved and used ethically before applying them to such environments.

Reproducibility of research. In an effort to ensure the reproducibility of our research, we have provided a detailed and transparent explanation of the research methodologies, and of the exact experiments conducted. Additionally, we strive to uphold the *FAIR (Findable, Accessible, Interoperable, Reusable)* principles. To this aim, all datasets used for training, all the results of the experiments and resulting figures have been made publicly available. The adapted code from the CORL library, the scripts used to generate the datasets and images, and the full code environment are also available. Running instructions enable reproducing the experiments described in this paper. These elements are available on GitHub. Finally, the hyper-parameters used, and the tuning methods are described in appendix C.

Data, bias, and plagiarism. Training RL models with data generated synthetically for the experiments, as is the case for our work, can be subject to bias. We have ensured to generate multiple datasets per policy, with logically expandable sequences of seeds (in increments of 50), to avoid manually selecting favorable seeds. Similarly, every other stochastic component was executed with different sequences of seeds. Additionally, neither the author nor the supervisors has had any affiliation or influence to or by a third party. Finally, to uphold academic integrity, we have ensured that all sources used in this work are fully and adequately documented.

References

[1] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996. [Online]. Available: https://doi.org/10.1613/jair.301

[2] Y. Li, "Reinforcement learning applications," *CoRR*, vol. abs/1908.06973, 2019. [Online]. Available: http://arxiv.org/abs/1908.06973

[3] Z. Ding and H. Dong, *Challenges of Reinforcement Learning*. Singapore: Springer Singapore, 2020, pp. 249–272. [Online]. Available: https://doi.org/10.1007/978-981-15-4095-0_7

[4] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *CoRR*, vol. abs/2005.01643, 2020. [Online]. Available: https://arxiv.org/abs/2005.01643

[5] I. Mediratta, Q. You, M. Jiang, and R. Raileanu, "The generalization gap in offline reinforcement learning," *CoRR*, vol. abs/2312.05742, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2312.05742

[6] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging procedural generation to benchmark reinforcement learning," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 2048–2056. [Online]. Available: http://proceedings.mlr.press/v119/cobbe20a.html

[7] A. Patterson, S. Neumann, M. White, and A. White, "Empirical design in reinforcement learning," *CoRR*, vol. abs/2304.01315, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2304.01315

[8] R. F. Prudencio, M. R. O. A. Maximo, and E. L. Colombini, "A survey on offline reinforcement learning: Taxonomy, review, and open problems," *IEEE Transactions on Neural Networks and Learning Systems*, 2023. [Online]. Available: https://doi.org/10.1109/TNNLS.2023.3250269

[9] I. Kostrikov, A. Nair, and S. 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. [Online]. Available: https://openreview.net/forum?id=68n2s9ZJWF8

[10] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4RL: datasets for deep data-driven reinforcement learning," *CoRR*, vol. abs/2004.07219, 2020. [Online]. Available: https://arxiv.org/abs/2004.07219

[11] H. He, C. Bai, K. Xu, Z. Yang, W. Zhang, D. Wang, B. Zhao, and X. Li, "Diffusion model is an effective planner and data synthesizer for multi-task reinforcement learning," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/ccda3c632cc8590ee60ca5ba226a4c30-Abstract-Conference.html

[12] P. Hansen-Estruch, I. Kostrikov, M. Janner, J. G. Kuba, and S. Levine, "IDQL: implicit q-learning as an actor-critic method with diffusion policies," *CoRR*, vol. abs/2304.10573, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2304.10573

[13] D. Pomerleau, "ALVINN: an autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems 1, [NIPS Conference, Denver, Colorado, USA, 1988]*, D. S. Touretzky, Ed. Morgan Kaufmann, 1988, pp. 305–313. [Online]. Available: http://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network

[14] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson, "Implicit behavioral cloning," in *Conference on Robot Learning, 8-11 November 2021, London, UK*, ser. Proceedings of Machine Learning Research, A. Faust, D. Hsu, and G. Neumann, Eds., vol. 164. PMLR, 2021, pp. 158–168. [Online]. Available: https://proceedings.mlr.press/v164/florence22a.html

[15] T. Seno and M. Imai, "d3rlpy: An offline deep reinforcement learning library," *J. Mach. Learn. Res.*, vol. 23, pp. 315:1–315:20, 2022. [Online]. Available: http://jmlr.org/papers/v23/22-0017.html

[16] D. Tarasov, A. Nikulin, D. Akimov, V. Kurenkov, and S. Kolesnikov, "CORL: research-oriented deep offline reinforcement learning library," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/62d2cec62b7fd46dd35fa8f2d4aeb52d-Abstract-Datasets_and_Benchmarks.html

[17] M. Weltevrede, M. T. J. Spaan, and W. Böhmer, "The role of diverse replay for generalisation in reinforcement learning," *CoRR*, vol. abs/2306.05727, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2306.05727

[18] M. Chevalier-Boisvert, B. Dai, M. Towers, R. Perez-Vicente, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry, "Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/e8916198466e8ef218a2185a491b49fa-Abstract-Datasets_and_Benchmarks.html

[19] K. Asadi and M. L. Littman, "An alternative softmax operator for reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 2017, pp. 243–252. [Online]. Available: http://proceedings.mlr.press/v70/asadi17a.html

[20] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, A. Teredesai, V. Kumar, Y. Li, R. Rosales,

E. Terzi, and G. Karypis, Eds. ACM, 2019, pp. 2623–2631. [Online]. Available: https://doi.org/10.1145/3292500.3330701

[21] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*. IEEE, 2012, pp. 5026–5033. [Online]. Available: https://doi.org/10.1109/IROS.2012.6386109

[22] V. Kurenkov, A. Nikulin, D. Tarasov, and S. Kolesnikov, "Katakomba: Tools and benchmarks for data-driven nethack," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/a5f596699d8d4637532f955c7f2860f4-Abstract-Datasets_and_Benchmarks.html

A Implementation details

In this appendix we present the most important implementation details for the IQL algorithm evaluated in this paper.

Adaptations for discrete control. IQL is an algorithm originally designed for continuous control, but our environment uses discrete control. The Katakomba library [22] offers an implementation for discrete control. However, this library is written specifically for the NetHack learning environment, therefore would need a reduction from the environment used in this paper to NetHack. Adapting an existing continuous control implementation was deemed to be a more reasonable approach. As such, the implementation of IQL from CORL [16] is adapated and used in this work. We have made the necessary changes to work within our discrete environment. First, the target network in a continuous setting receives both observations and actions as inputs, and outputs one estimation. However, in a discrete setting, we take observations as input and output an estimate for every action. This changes the target network $Q_{\hat{\theta}}(s, a)$ dimensionality to:

$$Q_{\hat{\theta}}(s) : \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|} \tag{8}$$

where $\mathcal{S}$ denotes the observation space and $\mathbb{R}^{|\mathcal{A}|}$ denotes a real valued vector of dimension equal to the size of the action space. This means that the target network will output an estimate for every action. Moreover, since we now have as many estimates as actions, we need to adapt the loss calculations for the value function $V_\psi(s)$:

$$L_V(\psi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\big[L_2^\tau(\text{gather}(Q_{\hat{\theta}}(s), a) - V_\psi(s))\big] \tag{9}$$

where "gather" is a function selecting the correct estimate from the target network along the action tensor (from `torch.gather`[1]). Using these adaptations, the rest of the loss calculations do not need any particular mathematical changes other than reshaping the inputs in some instances due to the changes in dimensionality specified in equations 8 and 9.

Network Architecture. The target network architecture is maintained from the CORL [16] library's implementation. Particularly, it uses a Multi-Layer Perceptron (MLP) with 2 hidden layers of dimension 256 each. Each hidden layer applies a linear transformation, followed by the activation function. Moreover, a twin Q-network is used to reduce over-estimation in the Q-function, by maintaining two MLP Q-value estimators Q_1, Q_2 and evaluating:

$$Q(s, a) = min(Q_1(s, a), Q_2(s, a)) \tag{10}$$

where Q is the final estimate for the Q-function.

Learned Policy. Within the adapatations for discrete control also comes changing the learned policy. The default CORL [16] learned policy is a Gaussian, which is of course not suited for our environment. Since we have a finite number of actions, we use a categorical policy instead. The policy network (also an MLP) outputs the mean logits for a categorical distribution over actions:

$$\pi(a|s) = \text{Categorical}(\text{logits} = MLP(s)) \tag{11}$$

[1] https://pytorch.org/docs/stable/generated/torch.gather.html

where $\pi(\cdot|s)$ is the learned policy. During evaluation, the action with the highest probability is picked:

$$a = \arg\max_a \pi(a|s) \tag{12}$$

B Environment details

This appendix provides the specific characteristics and observation space of the environment used in this paper. We use the environment from [17], which slightly changes the original implementation in Minigrid [18]. In essence, there are 4 rooms separated by walls, each wall having a doorway the agent can pass through to an adjacent room. The agent has a position and a direction and its objective is to reach the goal (a static object with a given position).

The action space is reduced to three discrete actions: turn left, right or go forward, and the reward function is simplified to be 1 when the agent reaches the goal and 0 otherwise. Moreover, the room size is reduced to 3x3 and the observation space is made fully observable. Particularly, observations consist of a 4x9x9 tensor centered around the agent's current location [17], and the 4 channels contain:

1. Agent position (center)
2. Agent's next position if we move forward with the current direction
3. Wall locations
4. Goal position

C Hyper-parameter tuning

Hyper-parameter tuning has been performed offline using an Optuna [20] study. Optuna studies provide the ability to use a series of sampling algorithms, among which we use the Tree-structured Parzen Estimator (TPE). The estimator is based on independent sampling, meaning that each parameter is chosen without considering relationships or correlation with other parameters. TPE on each trial fits, per parameter, one Gaussian Mixture Model (GMM) $\ell(x)$ to the set of parameter values that were used to obtain the best values for the objective function, and another GMM $g(x)$ to the rest of the parameter values. Then, it chooses a parameter value x such that the ratio $\frac{\ell(x)}{g(x)}$ is minimized.

The parameters considered for IQL are the learning rates for the target network, V-function, Q-function and actor, as well as the inverse temperature β and the coefficient for asymmetric loss τ. The learning rate for BC is also tuned. For the TPE estimator to know the search space for each parameter, they need to be specified. This is done by "suggestions", of which we use three types:

- *suggest_loguniform*: specifies a range of floating point numbers from which to sample uniformly
- *suggest_float*: specifies a finite list of floating point numbers, with a minimum and maximum, and a step size

- *suggest_int*: specifies a finite list of integers, with a minimum and maximum, and a step size

Each parameter mentioned above was given a range centered at the default values. These ranges are given in table 1.

Parameter	Suggestion
IQL V-network learning rate	$suggest\_loguniform(low = 10^{-4}, high = 10^{-2})$
IQL Q-network learning rate	$suggest\_loguniform(low = 10^{-4}, high = 10^{-2})$
IQL actor learning rate	$suggest\_loguniform(low = 10^{-4}, high = 10^{-2})$
IQL target network learning rate	$suggest\_float(low = 0.003, high = 0.007, step = 0.001)$
IQL coefficient for asymmetric loss	$suggest\_float(low = 0.5, high = 1.0, step = 0.1)$
IQL inverse temperature	$suggest\_float(low = 1.0, high = 3.0, step = 0.5)$
BC learning rate	$suggest\_loguniform(low = 10^{-4}, high = 10^{-2})$

Table 1: Suggestion type and range for every parameter considered during hyper-parameter tuning.

Note that BC has less hyper-parameters than IQL. To balance out the amount of tuning for each algorithm, we let each Optuna study run for the same amount of trials, i.e. the same amount of combinations of hyper-parameters will be tested for each algorithm. The final parameter values that we use are given in table 2.

Parameter	Value
IQL V-network learning rate	0.00030631780632380705
IQL Q-network learning rate	0.0001904542069504683
IQL actor learning rate	0.004450418567816549
IQL target network learning rate	0.006
IQL coefficient for asymmetric loss	0.8
IQL inverse temperature	3.0
BC learning rate	0.000701

Table 2: Final parameter values used.

Moreover, if we add the amount of training steps and random seed as hyper-parameters for each algorithm, we obtain the following results regarding their importance, in figures 8 and 9.

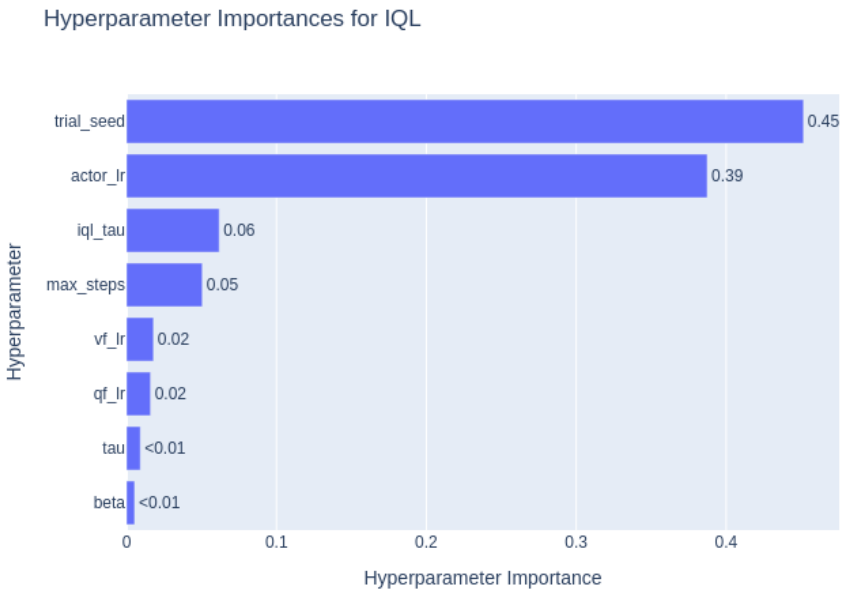


Figure 8: Relative hyper-parameter importance during tuning for IQL.

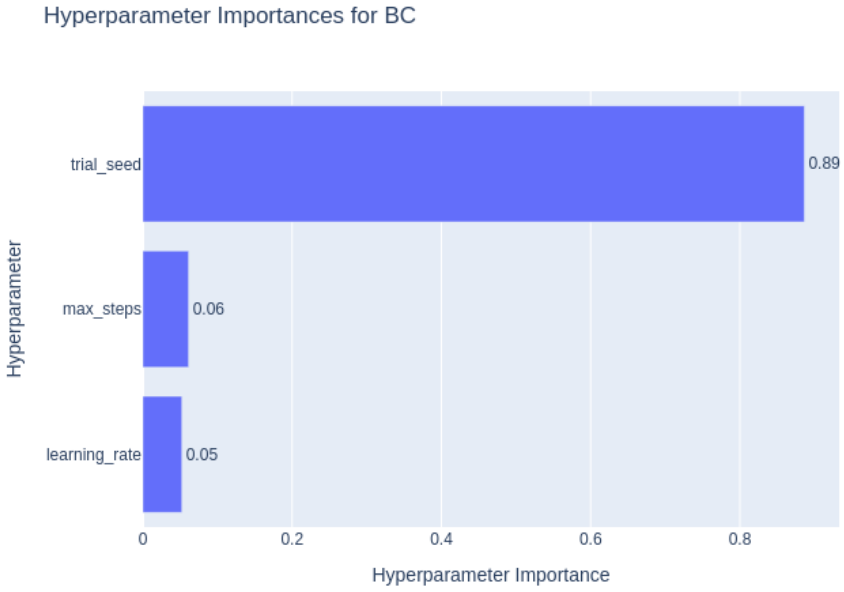


Figure 9: Relative hyper-parameter importance during tuning for BC.

D Full experimental results

For the sake of completeness, this appendix includes all training results for 100, 200, 500, 1000, 2000, 5000, 10000, 25000 and 50000 steps for the reachable (figures 14 through 18), and unreachable (figures 19 through 23) sets. The data for the expert, ϵ-greedy (best mixed) and fully random policies is summarized in figures 10, 11, 12, and 13.

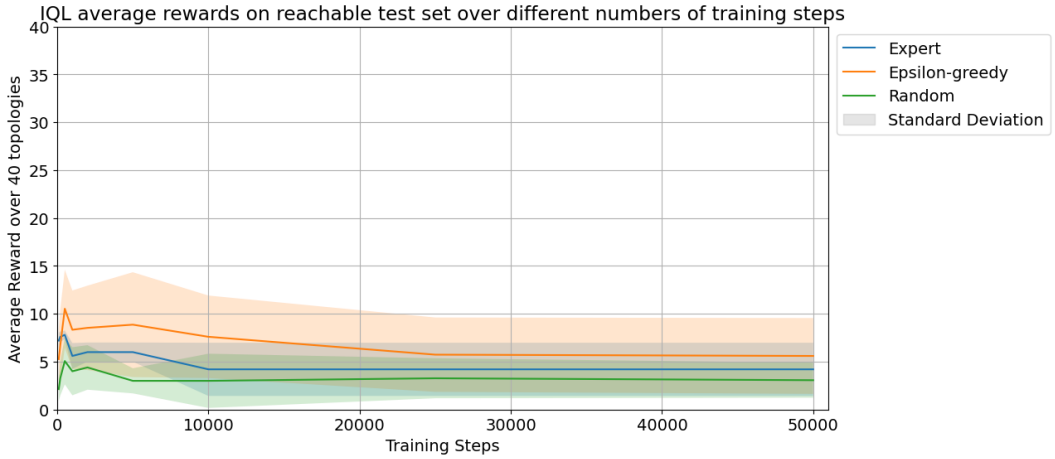


Figure 10: Average rewards obtained by IQL over different numbers of training steps for the different behavioral policies. Results averaged over 5 different seeds on the reachable topologies.

Figure 11: Average rewards obtained by BC over different numbers of training steps for the different behavioral policies. Results averaged over 5 different seeds on the reachable topologies.

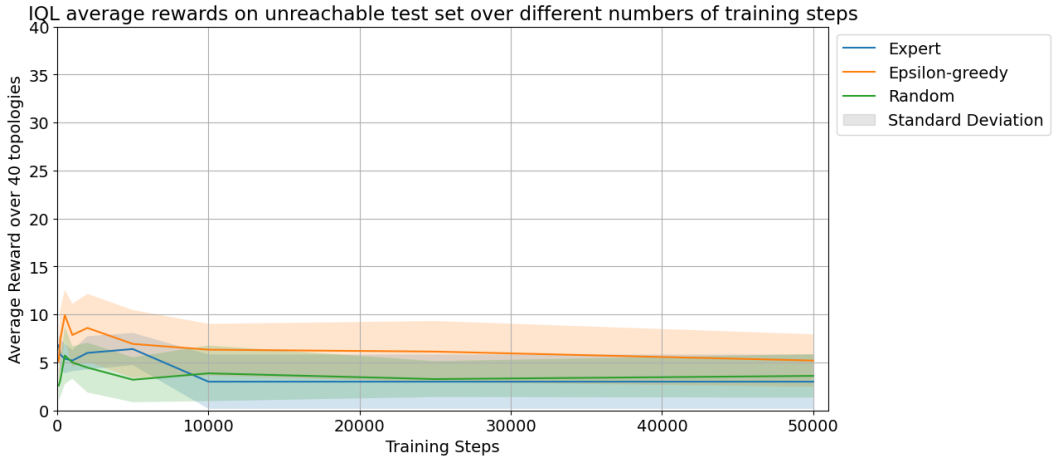


Figure 12: Average rewards obtained by IQL over different numbers of training steps for the different behavioral policies. Results averaged over 5 different seeds on the unreachable topologies.

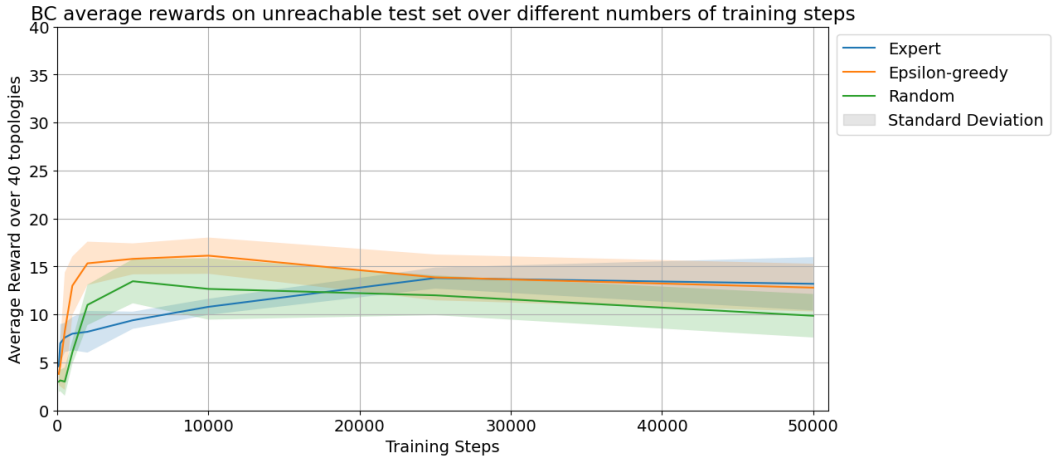
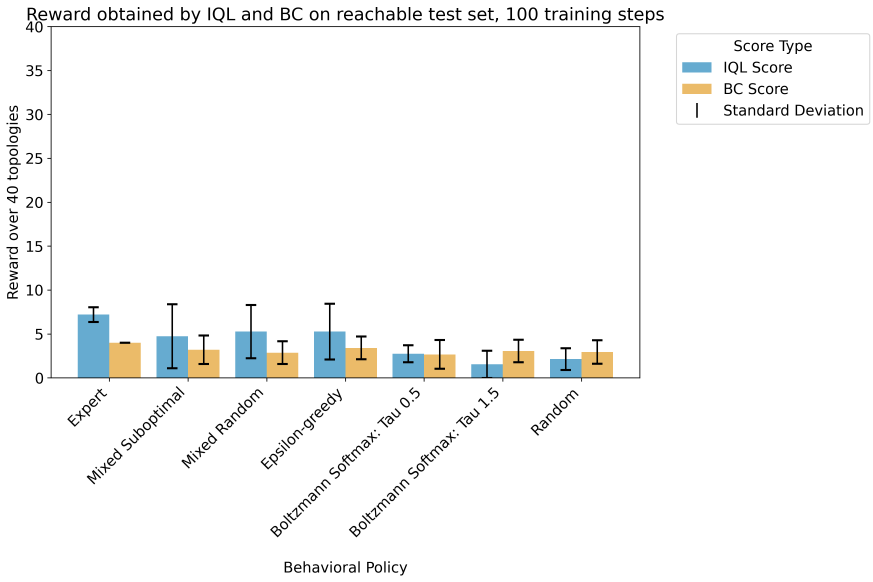
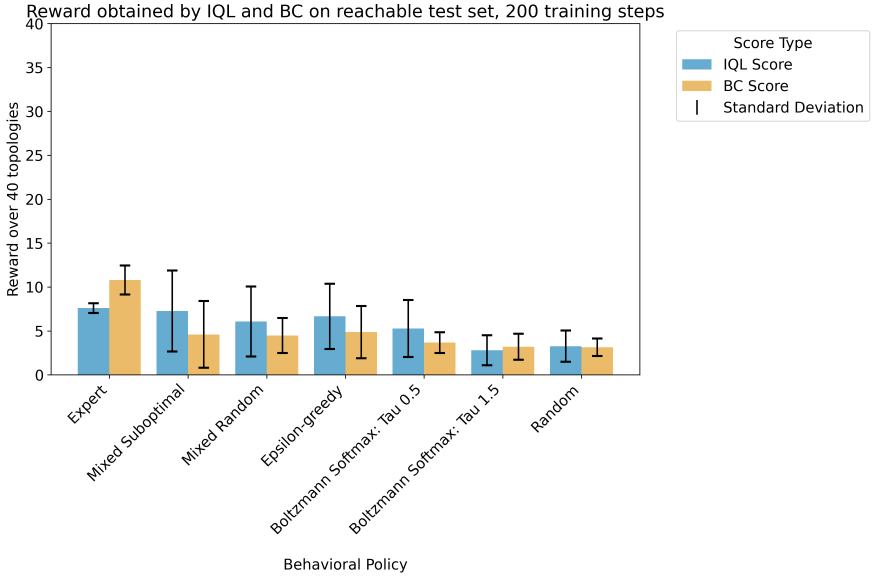


Figure 13: Average rewards obtained by BC over different numbers of training steps for the different behavioral policies. Results averaged over 5 different seeds on the unreachable topologies.

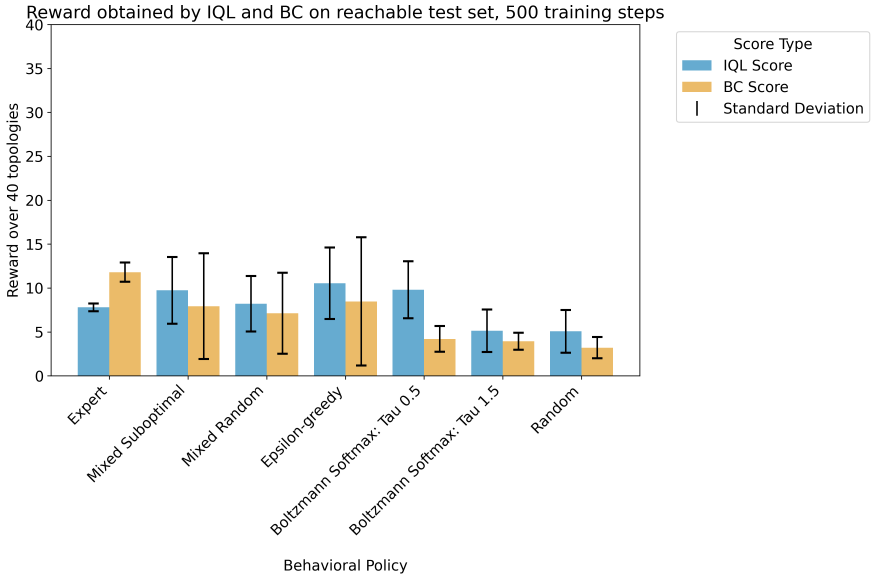


(a) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 reachable testing topologies and over 5 different seeds, trained for 100 steps on different dataset collection policies.

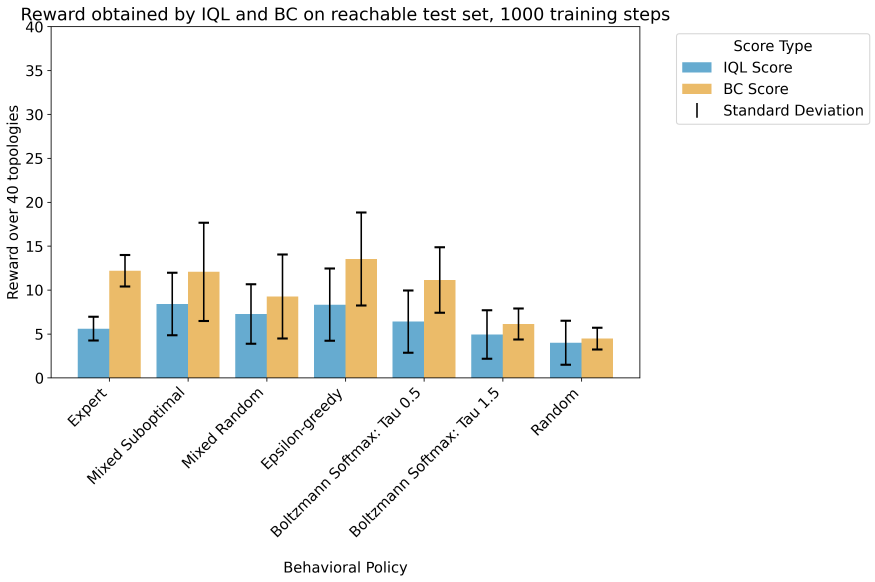


(b) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 reachable testing topologies and over 5 different seeds, trained for 200 steps on different dataset collection policies.

Figure 14: The average rewards obtained for reachable testing topologies, 100 and 200 steps.

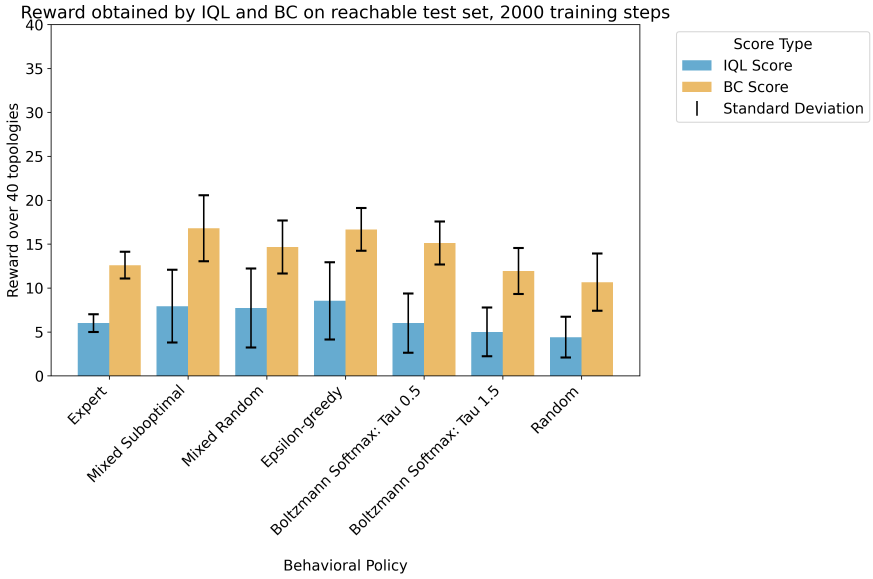


(a) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 reachable testing topologies and over 5 different seeds, trained for 500 steps on different dataset collection policies.

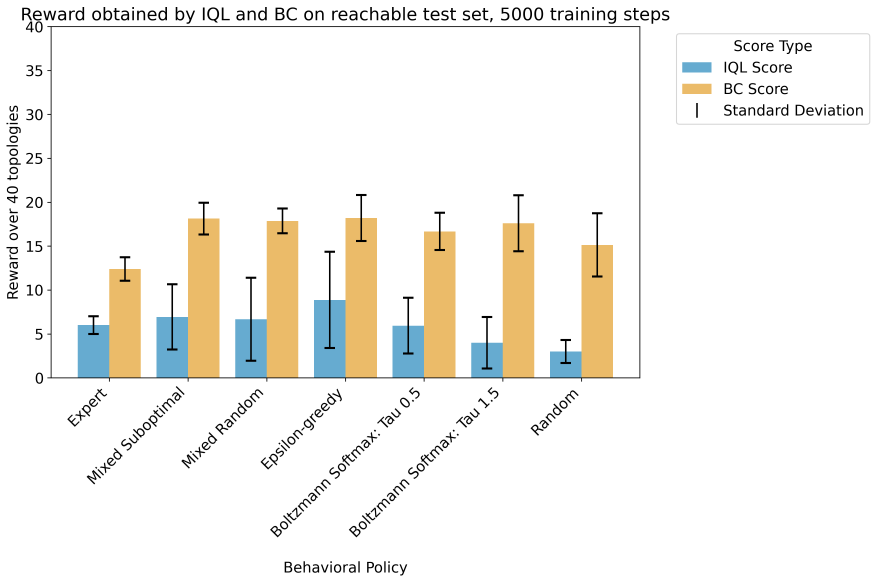


(b) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 reachable testing topologies and over 5 different seeds, trained for 1000 steps on different dataset collection policies.

Figure 15: The average rewards obtained for reachable testing topologies, 500 and 1000 steps.

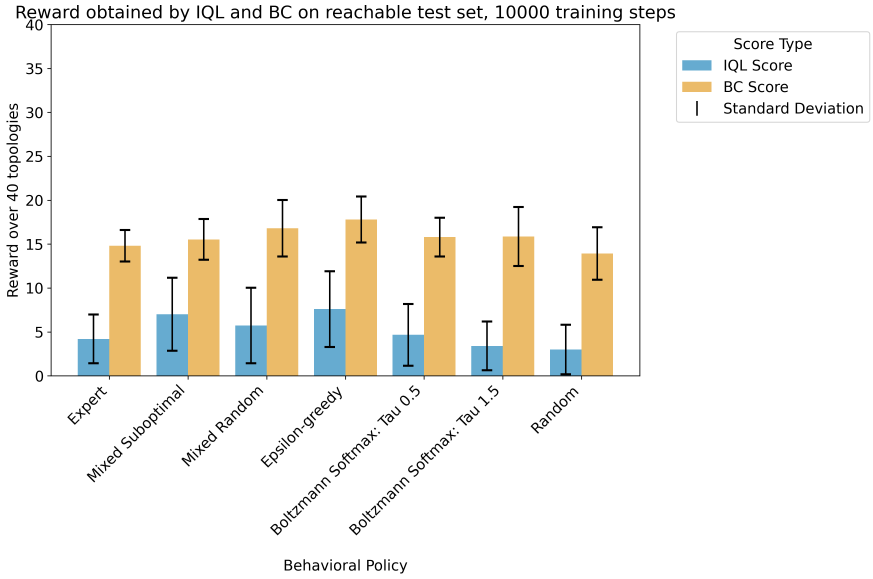


(a) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 reachable testing topologies and over 5 different seeds, trained for 2000 steps on different dataset collection policies.

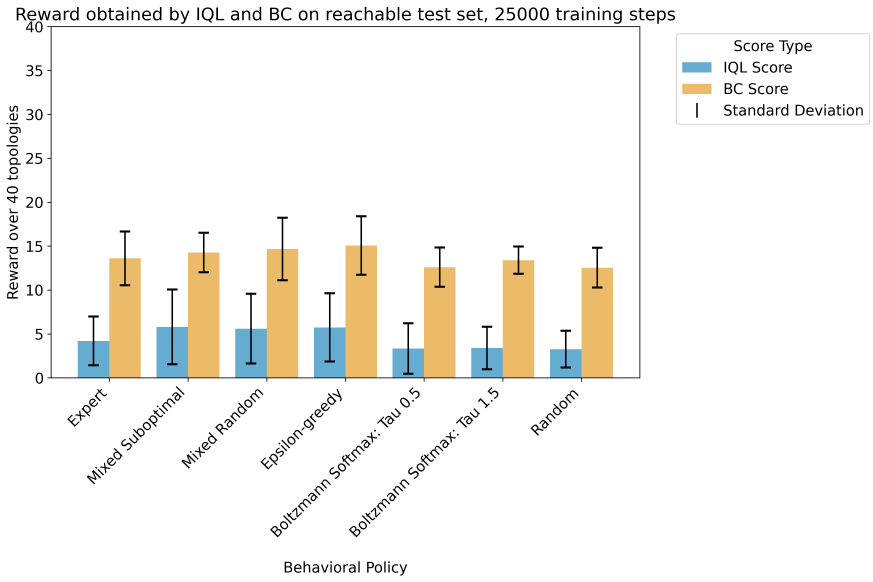


(b) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 reachable testing topologies and over 5 different seeds, trained for 5000 steps on different dataset collection policies.

Figure 16: The average rewards obtained for reachable testing topologies, 2000 and 5000 steps.



(a) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 reachable testing topologies and over 5 different seeds, trained for 10000 steps on different dataset collection policies.



(b) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 reachable testing topologies and over 5 different seeds, trained for 25000 steps on different dataset collection policies.

Figure 17: The average rewards obtained for reachable testing topologies, 10000 and 25000 steps.

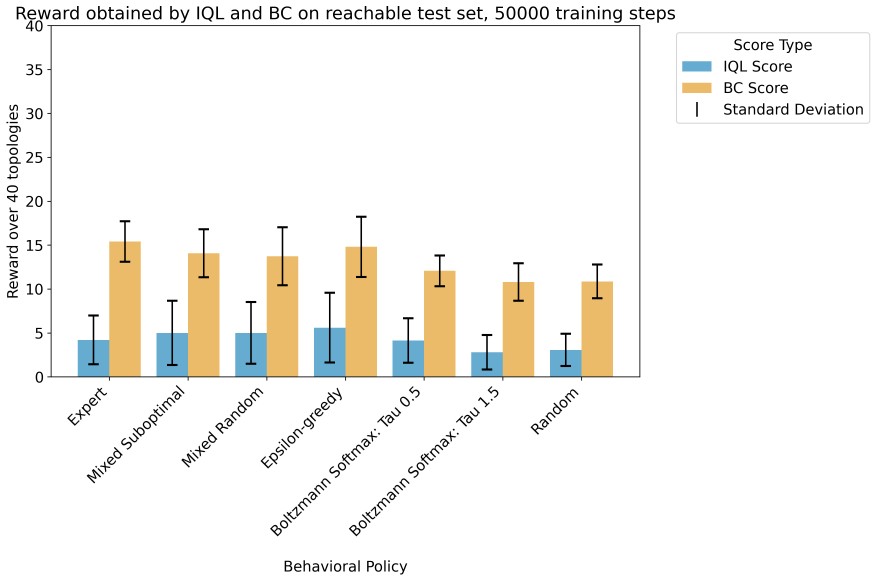
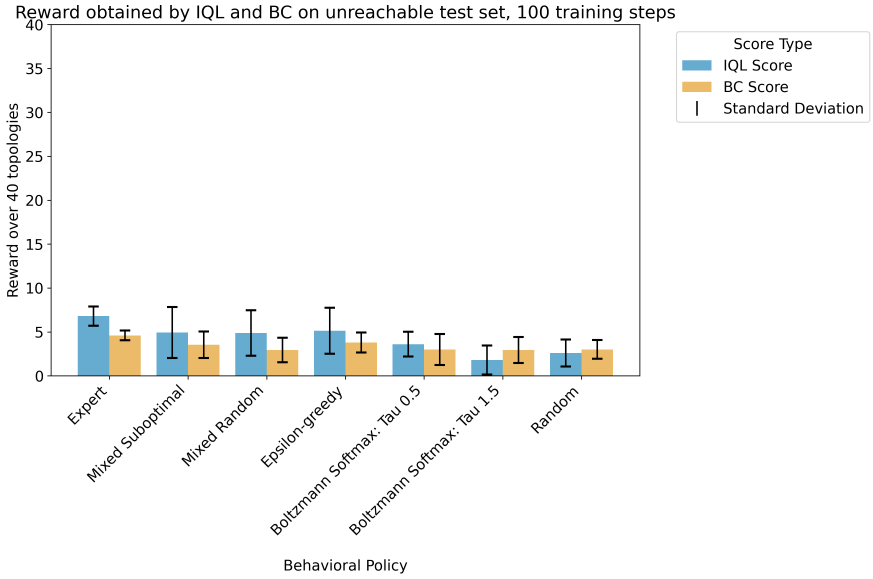
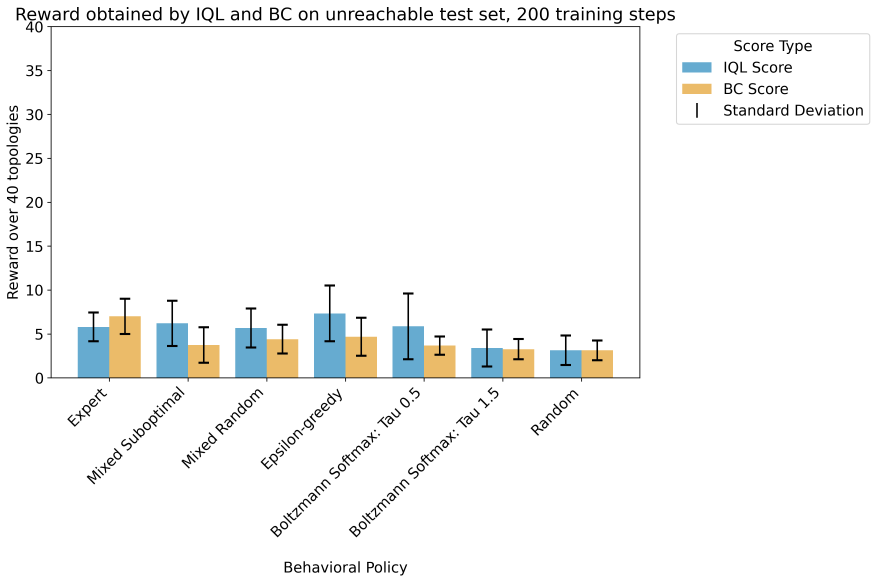


Figure 18: Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 reachable testing topologies and over 5 different seeds, trained for 50000 steps on different dataset collection policies.

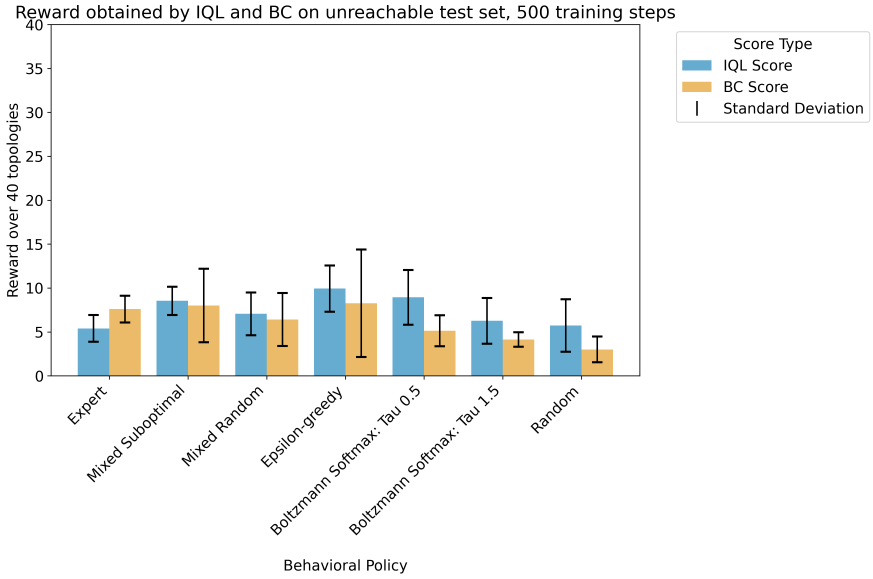


(a) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 unreachable testing topologies and over 5 different seeds, trained for 100 steps on different dataset collection policies.

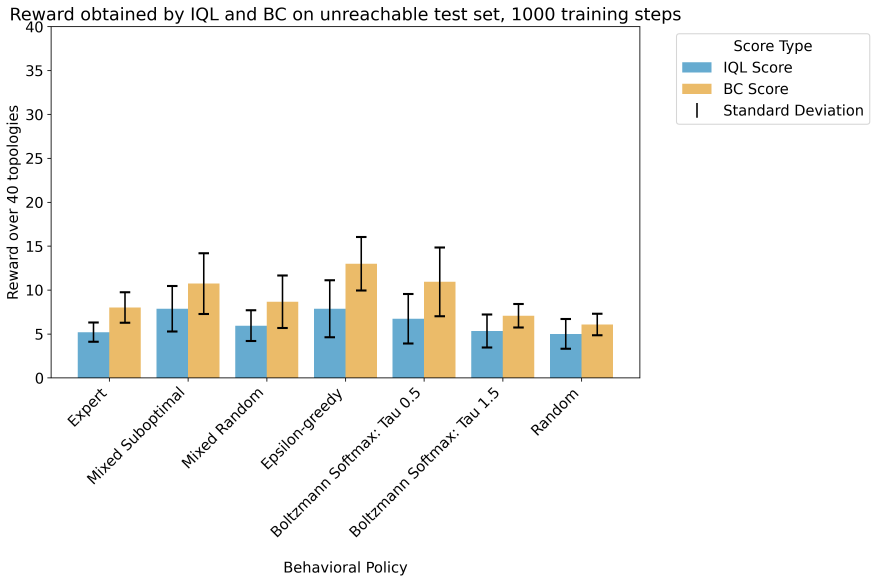


(b) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 unreachable testing topologies and over 5 different seeds, trained for 200 steps on different dataset collection policies.

Figure 19: The average rewards obtained for unreachable testing topologies, 100 and 200 steps.

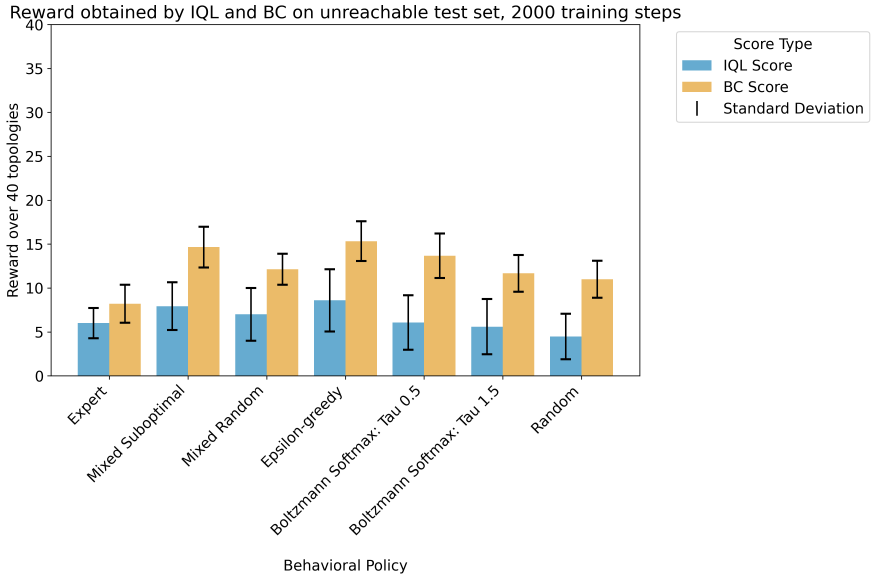


(a) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 unreachable testing topologies and over 5 different seeds, trained for 500 steps on different dataset collection policies.

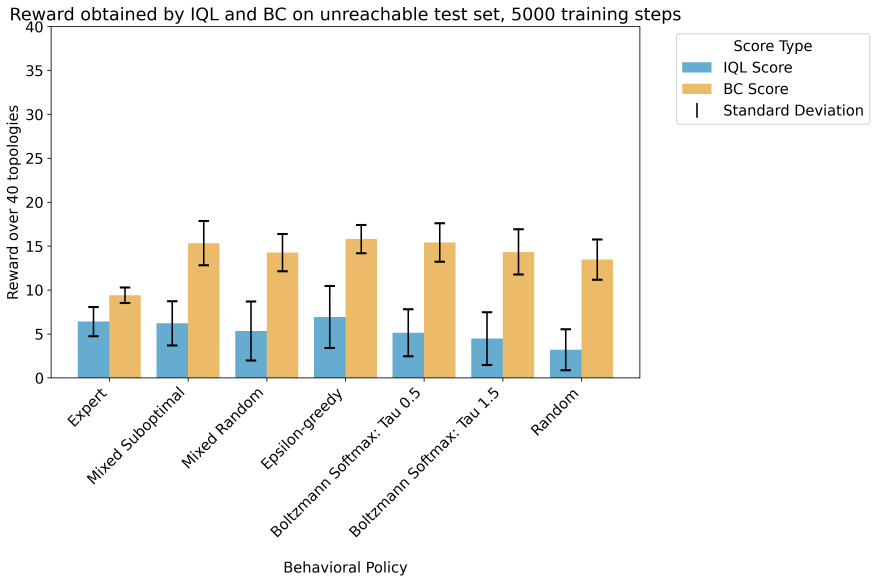


(b) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 unreachable testing topologies and over 5 different seeds, trained for 1000 steps on different dataset collection policies.

Figure 20: The average rewards obtained for unreachable testing topologies, 500 and 1000 steps.

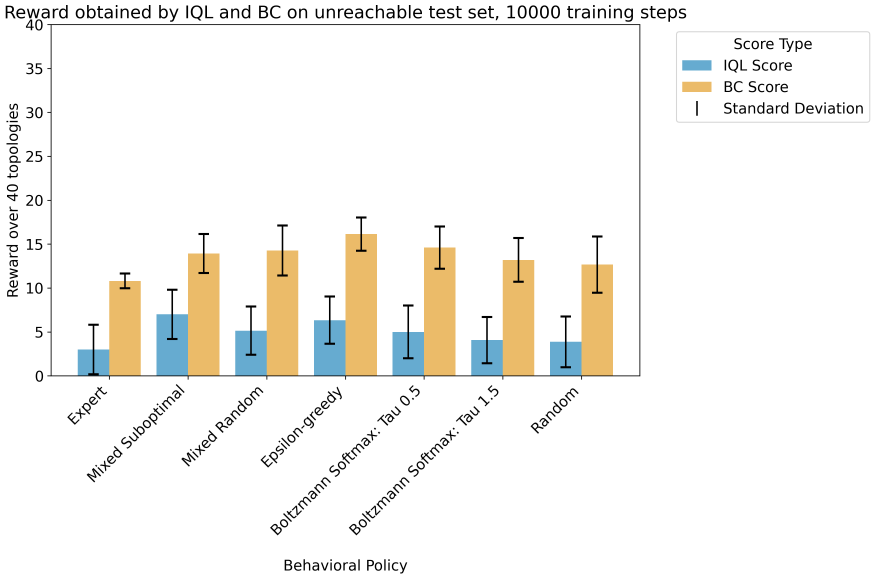


(a) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 unreachable testing topologies and over 5 different seeds, trained for 2000 steps on different dataset collection policies.

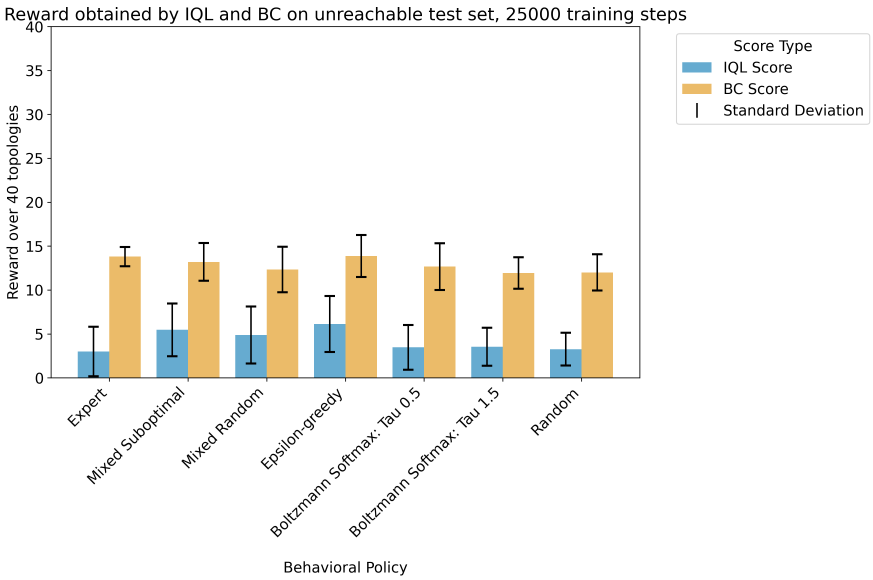


(b) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 unreachable testing topologies and over 5 different seeds, trained for 5000 steps on different dataset collection policies.

Figure 21: The average rewards obtained for unreachable testing topologies, 2000 and 5000 steps.



(a) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 unreachable testing topologies and over 5 different seeds, trained for 10000 steps on different dataset collection policies.



(b) Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 unreachable testing topologies and over 5 different seeds, trained for 25000 steps on different dataset collection policies.

Figure 22: The average rewards obtained for unreachable testing topologies, 10000 and 25000 steps.

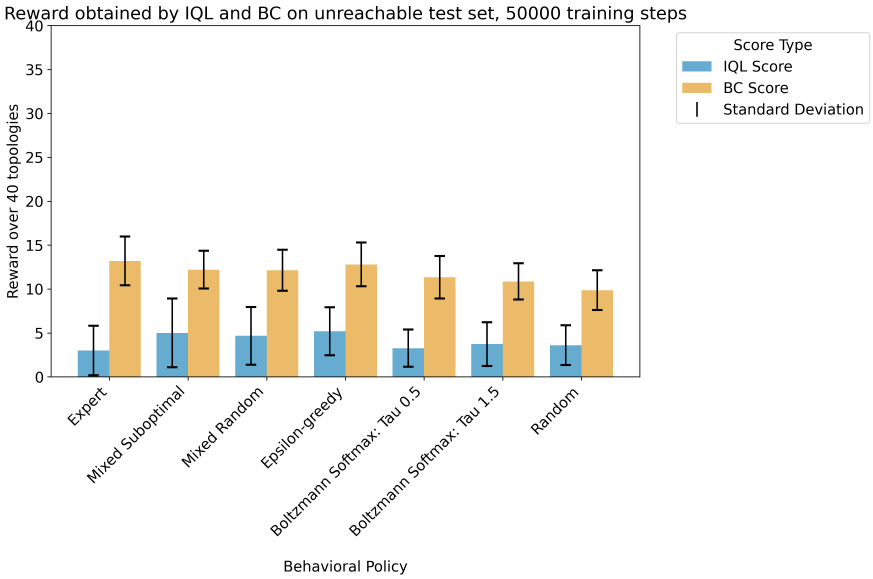


Figure 23: Average rewards and standard deviation obtained by IQL (blue) and BC (orange) over 40 unreachable testing topologies and over 5 different seeds, trained for 50000 steps on different dataset collection policies.