Understanding the Effects of Dataset Characteristics on Offline Reinforcement Learning

Kajetan Schweighofer [§ *] **Markus Hofmarcher** [§] **Marius-Constantin Dinu** [§, ‡]
Philipp Renz [§] **Angela Bitto-Nemling** [§] **Vihang Patil** [§] **Sepp Hochreiter** [§, †]

[§] ELLIS Unit Linz and LIT AI Lab,
Institute for Machine Learning,
Johannes Kepler University Linz, Austria
[‡] Dynatrace Research
[†] Institute of Advanced Research in Artificial Intelligence (IARAI)

Abstract

In real world, affecting the environment by a weak policy can be expensive or very risky, therefore hampers real world applications of reinforcement learning. Offline Reinforcement Learning (RL) can learn policies from a given dataset without interacting with the environment. However, the dataset is the only source of information for an Offline RL algorithm and determines the performance of the learned policy. We still lack studies on how dataset characteristics influence different Offline RL algorithms. Therefore, we conducted a comprehensive empirical analysis of how dataset characteristics effect the performance of Offline RL algorithms for discrete action environments. A dataset is characterized by two metrics: (1) the Trajectory Quality (TQ) measured by the average dataset return and (2) the State-Action Coverage (SACo) measured by the number of unique state-action pairs. We found that variants of the off-policy Deep Q-Network family require datasets with high SACo to perform well. Algorithms that constrain the learned policy towards the given dataset perform well for datasets with high TQ or SACo. For datasets with high TQ, Behavior Cloning outperforms or performs similarly to the best Offline RL algorithms.

1 Introduction

Central problems in Reinforcement Learning (RL) are credit assignment [2, 16, 27, 31] and efficiently exploring the environment [22]. Exploration in some problems can be costly because of high exploration or measurement costs, violating physical constraints, damaging the physical agent, costs of interaction with human experts, etc. [9]. In other cases exploration is risky, such as the risk of an accident for self-driving cars, the risk to crash the production machines if optimizing production processes, or the risk to loose money if applying RL in trading or pricing. In such cases, Offline Reinforcement Learning (RL), also referred to as Batch RL [21], offers to learn policies from pre-collected or logged dataset, without interacting with the environment [1, 12, 13, 20]. Offline Reinforcement Learning also avoids the need to build simulators, which are required for many tasks to train agents safely using Online RL [9]. Many such Offline RL datasets already exist for various real world problems [5, 8, 36]. Offline RL shares numerous traits with supervised deep learning, including, but not limited to leveraging large datasets. It has to face similar challenges such as generalization to unseen data, as stored samples may not cover the entire state-action space. In Offline RL, the generalization problem takes the form of distribution shift [30] during inference.

[*] Contact us at: kajetan.schweighofer@jku.at and patil@ml.jku.at

Workshop on Deep RL at NeurIPS 2021, Sydney, Australia.

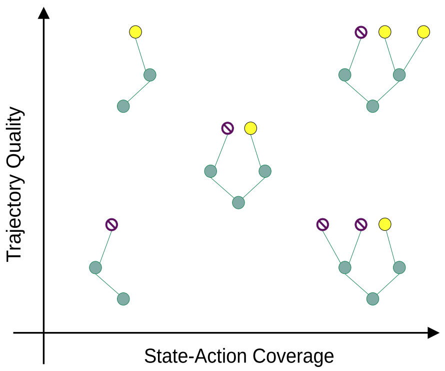

Figure 1: Trajectory Quality vs. State-Action Coverage of a given dataset. Datasets that are given as a set of trajectories are represented by graphs. Each graph represents all dataset trajectories, which start at the bottom root state and terminate at top leaf states. Edges represent actions. Green nodes are states with zero reward. Purple nodes are terminal states with low reward and yellow nodes are terminal states with high reward. Datasets are placed in this plot via the Trajectory Quality and State-Action Coverage. The performance of an Offline RL algorithm depends on the location of a dataset in this plot.

Multiple Offline RL algorithms [1, 12, 13, 15, 20, 34] have been proposed to address these problems and have shown good results. Well known off-policy algorithms such as Deep Q-Networks [23] can readily be used in Offline RL, by filling its replay-buffer with a pre-collected dataset. In practice, those algorithms often fail or lag far behind the performance they attain when trained in an Online RL setting. The reduced performance is attributed to the extrapolation errors for unseen state-action pairs and the distribution shift between the fixed given dataset and the states visited by the learned policy [12, 15]. Several algorithmic improvements tackle those problems, including policy constraints [12, 13, 34], regularization of learned action-values [20], and off-policy algorithms with more robust action-value estimates [1].

While unified datasets have been released [14, 11] for appropriate comparisons of Offline RL algorithms, we lack a proper understanding how the dataset characteristics influence the performance of different algorithms [29]. In this work, we therefore study this influence through the generation of datasets with different characteristics and compare the performance of Offline RL algorithms on these datasets. The characteristics of datasets depend on both the environment and the policy that generated them. To characterize datasets across environments and generating policies, we use two metrics: (1) the *Trajectory Quality* (TQ) measured by the average dataset return and (2) the *State-Action Coverage* (SACo) measured by the number of unique state-action pairs.

A dataset has high TQ, if its trajectories attain high rewards on average. A dataset has high SACo, if its trajectories cover a large proportion of all state-action pairs. Fig. 1 depicts datasets via these two metrics.

We conducted experiments on six different environments from three different environment suites [4, 6, 35], to create datasets with different characteristics (see Sec. 3.1). We executed 5,500 RL learning trials, which included a selection of algorithms [1, 7, 13, 15, 20, 23, 28, 34]. Then we analyzed their performance on datasets with different TQ and SACo. Variants of the off-policy Deep-Q-Network family [23, 1, 7] require datasets with high SACo to perform well. Algorithms that constrain the learned policy towards the given dataset perform well for datasets with high TQ or SACo. For datasets with high TQ, Behavior Cloning [28] gives better or equivalent performance compared to Offline RL algorithms.

2 Datasets for Offline Reinforcement Learning

We define our problem setting as a finite Markov decision process (MDP) to be a 5-tuple of $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$ of finite sets $\mathcal{S}$ with states s (random variable S_t at time t), $\mathcal{A}$ with actions a (random variable A_t), $\mathcal{R}$ with rewards r (random variable R_{t+1}), state-reward transition dynamics $p(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$, and $\gamma \in [0, 1)$ as a discount factor. The agent selects actions $a \sim \pi(S_t = s)$ based on the policy π which depends on the current state s. Our objective is to find the policy π which maximizes the expected return $G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1}$. In Offline RL, we assume that a dataset $\mathcal{D}$ of trajectories is provided. A single trajectory consists of a sequence of (s, a, r, s') tuples.

The dataset plays an important role in Offline RL. Access to diverse and sufficiently large datasets have been assumed in Offline RL literature [1, 13]. In most real world problems, dataset generation is not controlled by the practitioner. Some datasets could have more high return trajectories and may not cover the entire state-action space. While some may cover the entire state-action space but may contain less high return trajectories. Fig. 1 illustrates variants of this behavior. The characteristics of the dataset depends on data generation. As a result, algorithms behave differently for datasets generated in a different manner on the same problem.

Most publications use different dataset generation schemes for testing their Offline RL algorithms. [1] test on a dataset which consists of all training samples seen during training of a DQN agent. [12] generate data using a trained policy with an exploration factor and claim that this is close to a real world setting. [13] evaluate on multiple datasets which include a dataset comprising all training samples of a learning agent and data generated using a trained policy. [15] uses the RL Unplugged dataset [14] which comprises different datasets with different data generating regimes. Conservative Q-Learning [20] uses three datasets generated using a random, expert and a mixture of expert and random policy, generated from multiple different policies. [20] claim that data generated by multiple different policies fits a real world setting better, which conflicts with the claim made in [12]. Thus, there is an ambiguity in the Offline RL literature on what may be the correct data generation scheme to test Offline RL algorithms.

The performance of Conservative Q-Learning improved by changing dataset characteristics by modifying the dataset generation [20]. Similarly, in [15] data of high State-Action Coverage improved the performance of Offline RL algorithms have been compared to Behavior Cloning [28] with different dataset characteristics. These examples show that changing the dataset characteristics heavily influences the performance of Offline RL algorithms. But, this connection between dataset characteristics and the performance of Offline RL algorithms has not been explored in depth. Therefore, we conduct an experimental study on different dataset generation schemes used in Offline RL literature and investigate how the characteristics of a dataset affect the performance of Offline RL algorithms.

3 Study Design

In the next few sub-sections we outline the design of our study. To generate datasets with different characteristics, we introduce different dataset generation schemes (see 3.1). These generation schemes are related to existing schemes. Furthermore, we introduce measures for TQ and SACo (see Sec. 3.2) to assess the characteristics of different datasets. We describe the environments, algorithms, and the training parameters in detail.

3.1 Dataset Generation

We generate data in five different settings: 1) *random*, 2) *expert*, 3) *mixed* 4) *noisy* and 5) *replay*. For each of the datasets we have collected a predefined number of samples by interacting with the according environments (see Sec. 3.3). The number of samples in a dataset is determined by the number of environment interactions that are necessary to obtain expert policies through an Online RL algorithm. The baseline Online RL algorithm is Deep-Q-Network [23], which serves as an expert to create and collect samples under the different dataset characteristics as described below. Details on how the online policy was trained are given in the appendix (see Sec. A.4.2).

- **Random Dataset.** This dataset is generated using a fixed policy which selects random actions. Such a dataset was used for evaluation in [20]. It serves as a naive baseline on data collection.

- **Expert Dataset.** We trained an online policy until convergence and generated all samples with this final expert policy, without exploration. Such a dataset is used in [11, 15, 20].

- **Mixed Dataset.** The mixed dataset is generated using a mixture of the random dataset (20%) and the expert dataset (80%). This is similar to [11, 15] where they refer to such a dataset as *medium-expert*.
- **Noisy Dataset.** The noisy dataset is generated with an expert policy that selects the actions ϵ-greedy with $\epsilon = 0.2$. Creating a dataset from a fixed noisy policy is similar to the dataset creation process in [12, 13, 20, 15].
- **Replay Dataset.** This dataset is a collection of all samples generated by the online policy during training, thus multiple policies generated the data. This was used in [1, 13].

3.2 Evaluation Metrics

A dataset generating policy induces a state visitation distribution on the state space [32]. As a result, different generating policies result in different coverage of the state-action space. The coverage of the state-action space affects the performance of Offline RL algorithms [15]. To measure and compare quality and coverage properties of different datasets we use two metrics, the Trajectory Quality (TQ) and State-Action Coverage (SACo). Similar concepts have been introduced in [24], although no quantitative measures or further studies have been provided.

We define TQ as the average return of trajectories contained in the datasets compared to the maximal possible return. We define SACo as the ratio of the number of unique state-action pairs within each dataset and the number of all state-action pairs. To measure TQ and SACo, both the maximum achievable return and the entire state-action space is required. To acquire these information is infeasible for many environments, therefore we define relative measures that relate the TQ and SACo of each dataset to the online policy used to generate those datasets.

Relative Trajectory Quality (TQ). The relative TQ of a given dataset $\mathcal{D}$ is the normalized dataset return $g_{\mathcal{D}_{\text{norm}}}$ defined by

$$g_{\mathcal{D}_{\text{norm}}} = \frac{g_{\mathcal{D}} - g_{\text{min}}}{g_{\text{max}} - g_{\text{min}}}, \tag{1}$$

where $g_{\mathcal{D}}$ is the average return of the dataset.

The minimum return g_{min} is given as: $g_{\text{min}} = \text{minimum}(g_{\text{online}}, g_{\text{random}})$, where g_{online} is the maximum return achieved by the policy trained in an online fashion and g_{random} is the average return of the random policy. The maximum return g_{max} is: $g_{\text{max}} = \text{maximum}(g_{\text{online}}, g_{\text{random}})$. This is similar to the normalization done in [1] and is necessary as policies can perform worse than a random policy. Sec. A.6 (appendix) lists returns for each dataset and online policy.

Relative State-Action Coverage (SACo). The relative SACo $u_{\mathcal{D}_{\text{norm}}}$ of a dataset is defined as the ratio of unique state-action pairs in a dataset $u_{\mathcal{D}}$ and the unique state-action pairs of a reference dataset. We use the replay dataset u_{replay} as reference, since it was collected throughout training of the online policy and has a diverse set of state-actions pairs. Thus $u_{\mathcal{D}_{\text{norm}}}$ is,

$$u_{\mathcal{D}_{\text{norm}}} = \frac{u_{\mathcal{D}}}{u_{\text{replay}}}. \tag{2}$$

Counting unique state-action pairs of large datasets is often infeasible due to time and memory restrictions. Therefore, we used HyperLogLog [10] as a probabilistic counting method to determine the number of unique state-action pairs for each dataset. This ensures that the same evaluation procedure can be applied to large-scale benchmarks, such as the Arcade Learning Environment [3] (see Sec. A.5 in the appendix for details). The number of unique state-action pairs $u_{\mathcal{D}}$ and u_{replay} for the environments we consider are listed in Sec. A.6 in the appendix.

Different datasets could have the same relative SACo, even if these datasets consist of completely different trajectories. For example, a dataset with the same SACo as another dataset can have trajectories with much higher returns than trajectories from the second dataset. Therefore, SACo should be contrasted with other dataset metrics such as TQ.

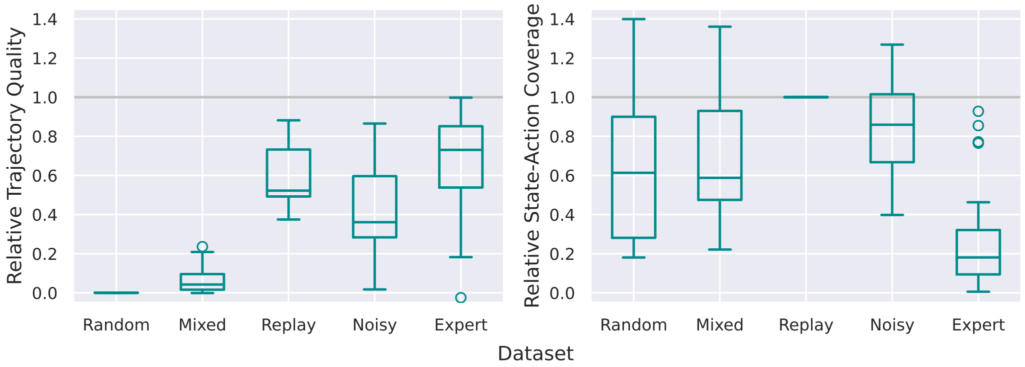

Figure 2: Relative TQ and Relative SACo over each dataset across dataset creation seeds and environments. The horizontal grey line (left) indicates the maximum return from an online policy. Replay dataset provides a good balance between TQ and SACo, which explains the good performance of most Offline RL algorithms using Replay dataset relative to other datasets.

3.3 Other Study Design Choices

We conduct the study on six different environments, from multiple suites. These are two classic control environments from the OpenAI gym suite [4], two MiniGrid [6] and two MinAtar environments [35]. For the first two suites, 10^5 samples were collected for every dataset, whereas $2 \cdot 10^6$ samples were collected for the MinAtar environments. Over all environments (six), different data generation schemes (five) and seeds (five), we generated a total number of 150 datasets.

We train nine different algorithms popular in the Offline RL literature including Behavior Cloning [28] and variants of Deep Q-Networks (DQN), Quantile-Regression DQN (QRDQN) [7] and Random Ensemble Mixture (REM) [1]. Furthermore, Behavior Value Estimation (BVE) [15] and Monte-Carlo Estimation (MCE) are used. Finally, three widely popular Offline RL algorithms, Batch-Constrained Q-learning (BCQ) [13], Conservative Q-learning (CQL) [19] and Critic Regularized Regression (CRR) [34] are considered. Details on specific implementations are given in the appendix in Sec. A.3.

The considered algorithms, were executed on each of the 150 datasets for five different seeds. Details on online and offline training are given in Sec. A.4. Experiments on MinAtar were only conducted for BC, DQN, BCQ and CQL due to computational constraints and are included in the results presented in Fig. 3. We study the performance of the policies trained using Offline algorithms relative to the trained online policy. The performance of the final policy p is given as follows:

$$p = \frac{g_{\text{offline}} - g_{\text{min}}}{g_{\text{max}} - g_{\text{min}}}. \tag{3}$$

Policies are evaluated in the environment after fixed intervals during offline training (see Sec. A.4). g_{offline} is the highest return of the policy during training averaged over training seeds. g_{min} and g_{max} are defined in Sec. 3.2. This results in all environments having similar range for performance score.

4 Analysis

We aim to analyse our experiments through the lens of dataset characteristics. Fig. 2 shows the relative TQ and the relative SACo of the gathered datasets, across dataset creation seeds and environments. Random and mixed datasets exhibit low relative TQ, while expert data has the highest TQ on average. On the other hand, expert data has low relative SACo on average, whereas random and mixed datasets are very diverse. The Replay dataset provides a good balance between TQ and SACo. Fig. A.1 in the appendix visualizes how generating the dataset influences the covered state-action space.

In Fig. 3, we plot the TQ and SACo of all generated datasets. Fig. 3 also shows the performance of each algorithm denoted by the color, on all generated datasets. These results indicate, that algorithms of the DQN family (DQN, QRDQN, REM) rely on high relative SACo to find a good policy. On the other end, BC works well only if datasets have high relative TQ, which is expected as its purpose is

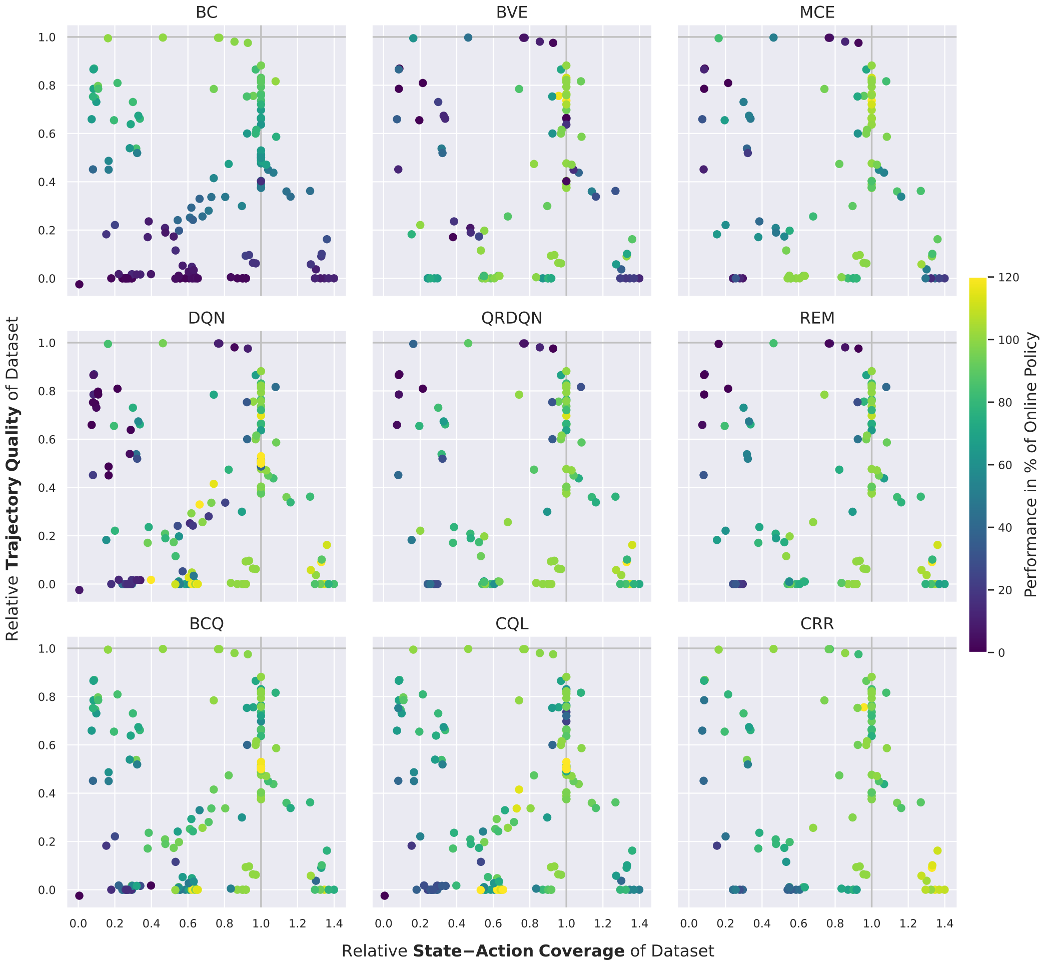

Figure 3: Each point is one of the 150 datasets and tested on each algorithm. We look at the Trajectory Quality (TQ) and State-Action Coverage (SACo) of each dataset, the color signifies the performance of an algorithm relative to the online policy. We see: a) BC improves as TQ increases b) DQN variants (middle row) require high SACo to do well c) Algorithms which constrain the policy towards data generating policy (bottom row) perform well across if datasets exhibit high TQ or SACo or both

to imitate behavior observed in the dataset. BVE and MCE were found to be very sensitive to the specific environment and dataset setting, favoring those with high relative SACo. BCQ, CQL and CRR enforce explicit or implicit constraints on the learned policy towards the behavioral policy and outperform algorithms of the DQN family, especially in those datasets with low relative SACo and high relative TQ. BCQ, CQL and CRR perform well if datasets exhibit high TQ or SACo or moderate values of TQ and SACo.

All the scores for all environments and algorithms over datasets are given in Sec. A.7. The scores in Sec. A.7 indicate that given a dataset from an expert, BC gives better or equivalent performance compared to Offline RL algorithms, despite not using any reward signal. When the data is not from an expert, Offline RL still performs well, while BC fails. The Replay dataset has relatively high TQ and SACo. Offline RL algorithms perform the best using the Replay dataset, compared to other datasets. Thus, it seems that there is more value in using Offline RL when data comes from multiple policies, which is the case in the Replay dataset.

Limitations. This work studies only the effects of the dataset for discrete action environments. The same comprehensive study has to be carried out on recently developed Offline RL algorithms for continuous control.

Conclusions. We conducted a comprehensive study of various Offline RL algorithms to understand the effect of dataset characteristics on their performance. Our study provides a blueprint for evaluating and understanding Offline RL algorithms in the future.

Acknowledgements. The ELLIS Unit Linz, the LIT AI Lab, the Institute for Machine Learning, are supported by the Federal State Upper Austria. IARAI is supported by Here Technologies. We thank the projects AI-MOTION (LIT-2018-6-YOU-212), DeepToxGen (LIT-2017-3-YOU-003), AI-SNN (LIT-2018-6-YOU-214), DeepFlood (LIT-2019-8-YOU-213), Medical Cognitive Computing Center (MC3), INCONTROL-RL (FFG-881064), PRIMAL (FFG-873979), S3AI (FFG-872172), DL for GranularFlow (FFG-871302), AIRI FG 9-N (FWF-36284, FWF-36235), ELISE (H2020-ICT-2019-3 ID: 951847), AIDD (MSCA-ITN-2020 ID: 956832). We thank Janssen Pharmaceutica (MaDeSMart, HBC.2018.2287), Audi.JKU Deep Learning Center, TGW LOGISTICS GROUP GMBH, Silicon Austria Labs (SAL), FILL Gesellschaft mbH, Anyline GmbH, Google, ZF Friedrichshafen AG, Robert Bosch GmbH, UCB Biopharma SRL, Merck Healthcare KGaA, Verbund AG, Software Competence Center Hagenberg GmbH, TÜV Austria, and the NVIDIA Corporation.

References

[1] R. Agarwal, D. Schuurmans, and M. Norouzi. An optimistic perspective on offline reinforcement learning. arXiv, 2020.

[2] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter. RUDDER: return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems 32*, pages 13566–13577, 2019.

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *ArXiv*, 2016.

[5] Serkan Cabi, Sergio Gómez Colmenarejo, Alexander Novikov, Ksenia Konyushkova, Scott E. Reed, Rae Jeong, Konrad Zolna, Yusuf Aytar, David Budden, Mel Vecerík, Oleg Sushkov, David Barker, Jonathan Scholz, Misha Denil, Nando de Freitas, and Ziyu Wang. A framework for data-driven robotics. *CoRR*, abs/1909.12200, 2019. URL http://arxiv.org/abs/1909.12200.

[6] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

[7] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression. arXiv, 2017.

[8] Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. Robonet: Large-scale multi-robot learning, 2020.

[9] Gabriel Dulac-Arnold, Daniel J. Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *CoRR*, abs/1904.12901, 2019. URL http://arxiv.org/abs/1904.12901.

[10] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *in aofa '07: proceedings of the 2007 international conference on analysis of algorithms*, 2007.

[11] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. arXiv, 2021.

[12] S. Fujimoto, E. Conti, M. Ghavamzadeh, and J. Pineau. Benchmarking batch deep reinforcement learning algorithms. arXiv, 2019.

[13] S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. arXiv, 2019.

[14] C. Gulcehre, Z. Wang, A. Novikov, T. Le Paine, S. G. Colmenarejo, K. Zolna, R. Agarwal, J. Merel, D. Mankowitz, C. Paduraru, G. Dulac-Arnold, J. Li, M. Norouzi, M. Hoffman, O. Nachum, G. Tucker, N. Heess, and N. de Freitas. Rl unplugged: Benchmarks for offline reinforcement learning. arXiv, 2020.

[15] C. Gulcehre, S. Gómez Colmenarejo, Z. Wang, J. Sygnowski, T. Paine, K. Zolna, Y. Chen, M. Hoffman, R. Pascanu, and N. de Freitas. Regularized behavior value estimation. arXiv, 2021.

[16] M. Holzleitner, L. Gruber, J. A. Arjona-Medina, J. Brandstetter, and S. Hochreiter. Convergence proof for actor-critic methods applied to PPO and RUDDER. *arXiv*, 2020.

[17] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9 (3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[18] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks, 2017.

[19] A. Kumar, J. Fu, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. arXiv, 2019.

[20] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. arXiv, 2020.

[21] Sascha Lange, Thomas Gabel, and Martin Riedmiller. *Batch Reinforcement Learning*, pages 45–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-27645-3. doi: 10.1007/978-3-642-27645-3_2. URL https://doi.org/10.1007/978-3-642-27645-3_2.

[22] R. McFarlane. A survey of exploration strategies in reinforcement learning. 2003.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing Atari with deep reinforcement learning. *ArXiv*, 2013.

[24] L. Monier, J. Kmec, A. Laterre, T. Pierrot, V. Courgeau, O. Sigaud, and K. Beguir. Offline reinforcement learning hands-on. *CoRR*, abs/2011.14379, 2020.

[25] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *ACM Queue*, (2):40–53, 4 2008.

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[27] V. P. Patil, M. Hofmarcher, M.-C. Dinu, M. Dorfer, P. M. Blies, J. Brandstetter, J. A. Arjona-Medina, and S. Hochreiter. Align-rudder: Learning from few demonstrations by reward redistribution. *CoRR*, abs/2009.14108, 2020.

[28] D. A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Comput.*, 3(1):88–97, 1991. ISSN 0899-7667.

[29] M. A. Riedmiller, J. T. Springenberg, R. Hafner, and N. Heess. Collect & infer - a fresh look at data-efficient reinforcement learning. *CoRR*, abs/2108.10273, 2021.

[30] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, 2010.

[31] R. S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci., 1984.

[32] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2 edition, 2018.

[33] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

[34] Z. Wang, A. Novikov, K. Zolna, J. T. Springenberg, S. Reed, B. Shahriari, N. Siegel, J. Merel, C. Gulcehre, N. Heess, and N. de Freitas. Critic regularized regression. arXiv, 2020.

[35] Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments, 2019.

[36] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning, 2020.

A Appendix

Contents of the appendix

List of figures

A.1 Introduction to the Appendix

This is the appendix to the paper "Understanding Dataset Generation for Offline RL". It provides more detailed information on the utilized environments, algorithms and gives more details on training of the online and offline policies.

A.2 Environments

While the dynamics of the introduced environments are rather different and range from motion equations to predefined game rules, they share common traits. This includes the dimension of the state $dim(\boldsymbol{s})$, the number of eligible actions $|\mathcal{A}|$, the maximum episode length T_{max} as well as the minimum and maximum expected return G_{min}, G_{max} of an episode. Furthermore, the discount factor γ is fixed for every environment regardless of the specific experiment executed on it and is thus listed with the other parameters in Tab. A.1.

Two environments contained in the MinAtar suite, Breakout and SpaceInvaders, do not have an explicit maximum episode length, as the episode termination is assured through the game rules. Breakout terminates either if the ball is hitting the ground or two rows of bricks were destroyed, which results in the maximum of 60 reward. An optimal agent could attain infinite reward for SpaceInvaders, as the aliens always reset if they are eliminated entirely by the player and there is a speed limit that aliens can maximally attain. Nevertheless, returns much higher than $200 - 300$ are very unlikely due to stochasticity in the environment dynamics that is introduced through sticky actions with a probability of 0.1 for all MinAtar environments.

Environment	$dim(\boldsymbol{s})$	$\|\mathcal{A}\|$	T_{max}	G_{min}	G_{max}	γ
`CartPole-v1`	4	2	500	9^*	500	0.95
`MountainCar-v0`	2	3	200	-200	-90^*	0.99
`MiniGrid-LavaGapS7-v0`	98	3	196	0	0.945^*	0.95
`MiniGrid-Dynamic-Obstacles-8x8-v0`	98	3	256	-1	0.935^*	0.95
`Breakout-MinAtar-v0`	100	3	-	0	60	0.99
`SpaceInvaders-MinAtar-v0`	100	4	-	0	∞	0.99

Table A.1: Environment specific characteristics and parameters. *Minimum or maximum expected returns depend on the starting state.

Action-spaces for MiniGrid and MinAtar are reduced to the number of eligible actions and state representations simplified. Specifically, the third layer in the state representation of MiniGrid environments was removed as it contained no information for the chosen environments. The state representation of MinAtar environments was collapsed into one layer, where the respective entries have been set to the index of the layer, divided by the total number of layers. The resulting two-dimensional state representations are flattened for MiniGrid as well as for MinAtar environments.

A.3 Algorithms

We conducted evaluation of the different dataset compositions using nine different algorithms applicable in an Offline RL setting. The selection covers recent advances in the field, as well as off-policy methods not specifically designed for Offline RL that are often utilized for comparison.

Behavioral cloning (BC) [28] serves as a baseline algorithm, as it mimics the behavioral policy used to create the dataset. Consequently, its performance is expected to be strongly correlated with the TQ of the dataset.

Behavior Value Estimation (BVE) [15] is utilized without the ranking regularization that it was proposed to be coupled with. This way, extrapolation errors are circumvented during training as the action-value of the behavioral policy is evaluated. Policy improvement only happens during inference, when the action is selected greedy on the learned action-values. BVE uses SARSA updates where the next state and action are sampled from the dataset, utilizing temporal difference updates to evaluate the policy.
As a comparison, Monte-Carlo Estimation (MCE) evaluates the behavioral policy that created the

dataset from the observed returns. Again, actions are selected greedy on the action-values obtained from Monte-Carlo estimates.

Deep Q-Network (DQN) [23] is used to obtain the online policy, but can be applied in the Offline RL setting as well, as it is an off-policy algorithm. The dataset serves as a replay buffer in this case, which remains constant throughout training. As it is not originally designed for the Offline RL setting, there are no countermeasures to the erroneous extrapolation of action-values during training nor during inference.
Quantile-Regression DQN (QRDQN) [7] approximates a set of K quantiles of the action-value distribution instead of a point estimate during training. During inference, the action is selected greedy through the mean values of the action-value distribution.
Random Ensemble Mixture (REM) [1] utilizes an ensemble of J action-value approximations to attain a more robust estimate. During training, the influence of each approximation on the overall loss is weighted through a randomly sampled categorical distribution. Selecting an action is done greedy on the average of the action-value estimates.

Batch-Constrained Deep Q-learning (BCQ) [12] for discrete action-spaces is based on DQN, but uses a BC policy on the dataset to constrain eligible actions during training and inference. A relative threshold τ is utilized for this constraint, where eligible actions must attain at least τ times the probability of the most probable action under the BC policy.
Conservative Q-learning (CQL) [20] introduces a regularization term to policy evaluation. The general framework might be applied to any off-policy algorithm that approximates action-values, therefore we based it on DQN as used for the online policy. Furthermore, the particular regularizer has to be chosen, where we used the KL-divergence against a uniform prior distribution, referred to as CQL($\mathcal{H}$) by the authors. The influence of the regularizing term is controlled by a temperature parameter α.
Critic Regularized Regression (CRR) [34] aims to ameliorate the problem that the performance of BC suffers from low-quality data, by filtering actions based on action-value estimates. Two filters which can be combined with several advantage functions were proposed by the authors, where the combination referred to as binary max was utilized in this study. Furthermore, DQN is used instead of a distributional action-value estimator for obtaining the m action-value samples in the advantage estimate.

A.4 Implementation Details

A.4.1 Network Architectures

The state input space is as defined in Tab. A.1, followed by 3 linear layers with a hidden size of 256. The number of output actions for the final linear layer is defined by the number of eligible actions for action-value networks. For QRDQN and REM, the number of actions times the number of quantiles or estimators respectively is used as output size. All except the last linear layer use the SELU activation function [18] with proper initialization of weights, whereas the final one applies a linear activation. Behavioral cloning networks use the softmax activation in the last layer to output a proper probability distribution, but are otherwise identical to the action-value networks.

A.4.2 Online Training

For every environment, a single online policy is obtained through training with DQN. This policy is the one used to generate the datasets under the different settings described in Sec. 3.1. All hyperparameters are listed in Tab. A.2.

Initially, as many samples as the batch size are collected by a random policy to pre-populate the experience replay buffer. Rather than training for a fixed amount of episodes, the number of policy-environment interactions is used as training steps. Consequently, the number of training steps is independent from the agents intermediate performance and comparable across environments. The policy is updated in every of those steps, after a single interaction with the environment where tuples $(\boldsymbol{s}, a, r, \boldsymbol{s}')$ are collected and stored in the buffer. After the buffer has reached the maximum size, the oldest tuple is discarded for every new one. Action selection during environment interactions to collect samples starts out with an initial ϵ that linearly decays over a period of steps towards the minimal ϵ, which remains fixed throughout the rest of the training procedure. Training batches are sampled randomly from the experience replay buffer. The Adam optimizer was used for all algorithms and the target network parameters θ' is updated to match the parameters θ of the current action-value estimator every 100 training steps.

The policy is evaluated periodically after a certain number of training steps, depending on the used environment. It interacts greedy based on the current value estimate with the environment for 10 episodes, averaging over the returns to estimate its performance.

Hyperparameter	Value
Algorithm	DQN
Learning rate	0.0001
Batch size	32
Optimizer	Adam
Loss	Huber with $\lambda = 1$
Initial ϵ	1.0
Linear ϵ decay period	1 000 steps
Minimal ϵ	0.01
Target update frequency	100 steps
Training steps	100 000 (2 000 000)
Network update frequency	1 step
Experience-Replay Buffer size	50 000 (500 000)
Evaluation frequency	200 (4 000) steps

Table A.2: Online training hyperparameters, values in parenthesis apply for MinAtar environments.

A.4.3 Offline Training

If not stated otherwise, the hyperparameters for offline training are identical to the ones used during online training, stated in Tab. A.2. All others which differ in an Offline RL setting are listed in Tab. A.3. Furthermore, parameters specific to the used algorithms are stated as well, relying on the parameters provided by the original authors.

Five times as many training steps as in the online case are used for training, which is common in Offline RL since one is interested in asymptotic performance on the fixed dataset. Algorithms are evaluated after a certain number of training steps through 10 interaction episodes with the environment, as it is done during the online training. Resulting returns for each of those evaluation steps are averaged over five independent runs, given an algorithm and a dataset. The maximum of this returns is then compared to the online policy through equation 3 to obtain the performance of the algorithm on a specific dataset.

Algorithm	Hyperparameter	Value
All	Evaluation frequency	1 000 (20 000) steps
All	Training steps	500 000 (10 000 000)
All	Batch size	128
QRDQN	Number of quantiles K	50
REM	Number of estimators J	200
BCQ	Threshold τ	0.3
CQL	Temperature parameter α	0.1
CRR	samples for advantage estimate m	4

Table A.3: Offline training hyperparameters, values in parenthesis apply for MinAtar environments.

A.4.4 Hardware and Software Specifications

Throughout the experiments, PyTorch 1.8 [26] with CUDA toolkit 11 [25] on Python 3.8 [33] was used. Plots are created using Matplotlib 3.4 [17].

We used a mixture of 27 GPUs, including GTX 1080 Ti, TITAN X, and TITAN V. Runs for Classic Control and MiniGrid environments took 96 hours in total, the executed runs for MinAtar environments took around 10 days.

A.5 Counting Unique State-Action Pairs

Due to time and memory restrictions, we evaluated several methods to enable counting on large benchmark datasets. We compared a simple list based approach to store all state-action pairs, a Hash-Table and the probabilistic counting method HyperLogLog [10]. We used the HyperLogLog approach, because it approximates the probability of obtaining certain properties in hashes created from the state-action pairs. HyperLogLog has a worst case time complexity of $\mathcal{O}(N)$ and worst case memory complexity of $\Theta(\log \log N)$, as there is no need to store a list of unique values. Even for large $N > 10^9$, estimates typically deviate by a maximum of 2% from the true counts as proven in [10]. We provide an overview of the time and memory complexities of all methods in Tab. A.4.

Algorithm	Time complexity	Memory complexity
List of uniques	$\mathcal{O}(N^2)$	$\Theta(N)$
Hash Table	$\mathcal{O}(N)$	$\Theta(N)$
HyperLogLog	$\mathcal{O}(N)$	$\Theta(\log \log N)$

Table A.4: Time and Memory complexities of different algorithms that count unique state-action pairs.

A.6 Calculating Relative TQ and SACo

All necessary measurements for calculating the relative TQ and SACo are listed in this section. The maximum returns attained by the online policy are listed in Tab. A.5, the average return attained by the random policy in Tab. A.6. Furthermore, the maximum return and unique state-action pairs of each dataset are given in Tab. A.7 and Tab. A.8.

Environment	Maximum return of online policy g_{online}				
	Run 1	Run 2	Run 3	Run 4	Run 5
CartPole-v1	500.00	500.00	500.00	500.00	500.00
MountainCar-v0	-99.78	-102.07	-102.70	-100.19	-99.82
MiniGrid-LavaGapS7-v0	0.80	0.91	0.86	0.81	0.85
MiniGrid-Dynamic-Obstacles-8x8-v0	0.93	0.93	0.93	0.93	0.92
Breakout-MinAtar-v0	18.02	19.46	17.00	18.47	19.32
SpaceInvaders-MinAtar-v0	26.31	25.17	28.45	28.09	28.08

Table A.5: Maximum return of the policy trained online.

Environment	Average return of the random policy g_{random}				
	Run 1	Run 2	Run 3	Run 4	Run 5
CartPole-v1	22.23	22.12	22.04	22.51	22.05
MountainCar-v0	-200.00	-200.00	-200.00	-200.00	-200.00
MiniGrid-LavaGapS7-v0	0.02	0.02	0.02	0.02	0.03
MiniGrid-Dynamic-Obstacles-8x8-v0	-1.00	-1.00	-1.00	-1.00	-1.00
Breakout-MinAtar-v0	0.51	0.51	0.51	0.51	0.51
SpaceInvaders-MinAtar-v0	2.84	2.83	2.84	2.85	2.85

Table A.6: Average return of the random policy.

Environment	Dataset	Average return of dataset trajectories $g_{\mathcal{D}}$				
		Run 1	Run 2	Run 3	Run 4	Run 5
`CartPole-v1`	Random	22.23	22.12	22.04	22.51	22.05
	Mixed	27.47	27.15	26.79	26.87	26.17
	Replay	208.05	249.72	201.13	215.27	201.98
	Noisy	397.03	144.62	248.48	116.77	77.21
	Expert	497.48	498.82	279.08	127.98	109.23
`MountainCar-v0`	Random	-200.00	-200.00	-200.00	-200.00	-200.00
	Mixed	-176.36	-183.04	-179.71	-182.96	-181.01
	Replay	-159.69	-135.38	-135.44	-133.67	-136.20
	Noisy	-156.13	-164.55	-164.98	-155.13	-166.10
	Expert	-118.90	-135.23	-128.93	-134.63	-132.52
`MiniGrid-LavaGapS7-v0`	Random	0.02	0.02	0.02	0.02	0.03
	Mixed	0.09	0.05	0.16	0.10	0.08
	Replay	0.59	0.70	0.71	0.57	0.62
	Noisy	0.61	0.56	0.70	0.70	0.65
	Expert	0.63	0.42	0.75	0.70	0.57
`MiniGrid-Dynamic-Obstacles-8x8-v0`	Random	-1.00	-1.00	-1.00	-1.00	-1.00
	Mixed	-0.87	-0.88	-0.82	-0.81	-0.99
	Replay	0.58	0.71	0.53	0.57	0.46
	Noisy	-0.09	0.14	0.16	0.19	-0.42
	Expert	0.89	0.89	0.92	0.93	0.00
`Breakout-MinAtar-v0`	Random	0.51	0.51	0.51	0.51	0.51
	Mixed	0.80	0.81	0.80	0.80	0.81
	Replay	9.53	10.04	8.92	9.25	9.72
	Noisy	4.91	6.90	4.48	4.86	5.78
	Expert	13.59	15.61	12.56	14.02	15.28
`SpaceInvaders-MinAtar-v0`	Random	2.84	2.83	2.84	2.85	2.85
	Mixed	4.07	2.81	4.07	3.54	3.71
	Replay	14.85	14.66	15.62	15.46	15.48
	Noisy	9.71	3.22	11.46	11.16	13.32
	Expert	14.26	2.28	16.65	14.21	18.96

Table A.7: Average return of dataset trajectories per environment and dataset creation setting for every run.

Environment	Dataset	Unique state-action pairs of environment $u_{\mathcal{D}}$				
		Run 1	Run 2	Run 3	Run 4	Run 5
`CartPole-v1`	Random	55 916	52 888	58 127	52 100	54 085
	Mixed	52 409	59 350	60 820	52 896	53 467
	Replay	95 384	94 749	95 950	96 499	97 263
	Noisy	70 710	64 392	78 952	53 173	51 771
	Expert	15 496	43 860	30 434	19 349	14 909
`MountainCar-v0`	Random	3 315	3 294	3 448	3 015	3 212
	Mixed	5 294	5 838	5 891	4 725	5 980
	Replay	13 740	11 183	12 411	12 444	12 549
	Noisy	14 669	14 187	14 138	12 934	14 575
	Expert	2 947	3 768	3 709	2 432	4 123
`MiniGrid-LavaGapS7-v0`	Random	1 842	1 847	1 879	1 919	1 840
	Mixed	1 819	1 813	1 827	1 866	1 808
	Replay	1 368	1 394	1 343	1 401	1 421
	Noisy	1 310	1 288	1 450	1 360	1 311
	Expert	114	112	116	116	104
`MiniGrid-Dynamic-Obstacles-8x8-v0`	Random	41 497	40 791	40 843	41 591	41 110
	Mixed	43 278	44 118	42 968	43 164	37 401
	Replay	45 283	45 423	46 916	46 191	44 801
	Noisy	46 571	49 191	45 526	44 998	40 115
	Expert	38 704	42 140	36 202	35 331	14 435
`Breakout-MinAtar-v0`	Random	16 218	15 915	16 459	16 247	16 182
	Mixed	18 351	18 608	20 175	18 179	19 166
	Replay	62 737	54 810	91 183	61 433	59 980
	Noisy	38 326	44 074	49 592	38 527	42 789
	Expert	5 809	5 914	9 006	4 950	6 535
`SpaceInvaders-MinAtar-v0`	Random	935 920	920 093	925 641	934 557	933 024
	Mixed	860 935	777 601	898 787	869 611	901 127
	Replay	1 507 798	1 463 980	1 446 246	1 439 305	1 426 702
	Noisy	933 016	582 548	1 053 096	955 208	1 057 379
	Expert	250 085	8 163	407 306	239 007	409 359

Table A.8: Unique state-action pairs per environment and dataset creation setting for every run.

A.7 Performance of Offline Algorithms

Performances as fraction of the respective online policy for every algorithm with the respective dataset settings are given in Tab. A.9 and Tab. A.10. The results pose averages over the different dataset creation seeds and multiple runs carried out with each algorithm, compared to the respective online policy used to create the dataset.

Dataset	BC	BVE	MCE	DQN	QRDQN	REM	BCQ	CQL	CRR
`CartPole-v1`									
Random	0.01 ±0.00	**0.96** ±0.02	**0.99** ±0.01	0.61 ±0.06	0.79 ±0.14	0.83 ±0.06	0.59 ±0.07	0.65 ±0.06	0.38 ±0.03
Mixed	0.08 ±0.04	0.86 ±0.17	**0.99** ±0.02	0.67 ±0.10	0.81 ±0.12	0.73 ±0.09	0.78 ±0.12	0.73 ±0.15	0.63 ±0.19
Replay	0.54 ±0.04	**1.00** ±0.00	**0.97** ±0.04	**0.97** ±0.04	**1.00** ±0.00	**1.00** ±0.00	**0.99** ±0.01	**0.98** ±0.02	**0.98** ±0.01
Noisy	0.51 ±0.28	**0.95** ±0.06	0.88 ±0.08	0.79 ±0.12	**0.96** ±0.05	**0.95** ±0.10	0.82 ±0.30	0.82 ±0.28	0.87 ±0.13
Expert	**0.65** ±0.36	**0.67** ±0.20	**0.59** ±0.15	**0.73** ±0.18	**0.71** ±0.24	0.53 ±0.29	**0.68** ±0.37	**0.71** ±0.31	**0.72** ±0.33
`MountainCar-v0`									
Random	0.00 ±0.00	**0.70** ±0.03	0.14 ±0.09	0.42 ±0.06	0.35 ±0.06	0.26 ±0.06	0.42 ±0.06	0.35 ±0.07	0.41 ±0.07
Mixed	0.10 ±0.02	0.13 ±0.08	0.52 ±0.07	**0.85** ±0.07	0.80 ±0.02	0.73 ±0.04	**0.90** ±0.03	0.79 ±0.03	0.78 ±0.03
Replay	0.56 ±0.20	0.15 ±0.15	**0.99** ±0.07	0.86 ±0.11	0.86 ±0.10	0.81 ±0.10	0.85 ±0.05	**0.92** ±0.03	0.85 ±0.05
Noisy	0.46 ±0.03	0.33 ±0.12	0.67 ±0.15	**0.86** ±0.06	0.79 ±0.05	0.80 ±0.09	0.81 ±0.05	**0.91** ±0.04	**0.86** ±0.06
Expert	**0.79** ±0.05	0.10 ±0.09	0.43 ±0.23	0.58 ±0.30	0.61 ±0.31	0.54 ±0.29	**0.80** ±0.04	**0.78** ±0.07	**0.78** ±0.06
`MiniGrid-LavaGapS7-v0`									
Random	0.11 ±0.01	0.20 ±0.03	0.30 ±0.13	0.91 ±0.12	0.86 ±0.14	0.80 ±0.13	0.88 ±0.14	0.63 ±0.14	**1.09** ±0.05
Mixed	0.21 ±0.06	0.72 ±0.22	0.85 ±0.18	**1.06** ±0.11	1.03 ±0.13	1.04 ±0.13	0.75 ±0.20	0.65 ±0.13	**1.11** ±0.05
Replay	0.81 ±0.08	**1.07** ±0.08	**1.10** ±0.06	0.91 ±0.13	0.90 ±0.12	0.90 ±0.13	0.71 ±0.07	0.43 ±0.17	0.85 ±0.04
Noisy	**0.85** ±0.12	0.82 ±0.21	0.76 ±0.10	0.57 ±0.25	0.51 ±0.26	0.53 ±0.18	0.66 ±0.13	0.65 ±0.13	**0.97** ±0.12
Expert	**0.65** ±0.13	0.17 ±0.19	0.10 ±0.12	0.12 ±0.09	0.10 ±0.15	0.08 ±0.13	**0.65** ±0.13	**0.65** ±0.13	**0.60** ±0.21
`MiniGrid-Dynamic-Obstacles-8x8-v0`									
Random	0.02 ±0.01	0.73 ±0.06	0.80 ±0.03	**1.00** ±0.00	**1.00** ±0.00	**1.00** ±0.00	**1.00** ±0.00	0.88 ±0.04	0.72 ±0.04
Mixed	0.17 ±0.07	**1.00** ±0.01	**0.99** ±0.01	**1.00** ±0.00	**1.00** ±0.00	**0.98** ±0.05	0.90 ±0.19	0.90 ±0.19	0.94 ±0.11
Replay	0.90 ±0.04	**1.00** ±0.00	**1.00** ±0.00	**0.99** ±0.01	**1.00** ±0.01	**1.00** ±0.01	**0.99** ±0.01	**0.98** ±0.02	**0.99** ±0.01
Noisy	0.71 ±0.09	**0.96** ±0.02	**0.95** ±0.03	0.90 ±0.11	0.89 ±0.14	0.90 ±0.13	0.92 ±0.15	0.93 ±0.14	**0.99** ±0.01
Expert	**0.90** ±0.19	0.14 ±0.14	0.10 ±0.07	0.12 ±0.09	0.11 ±0.10	0.14 ±0.19	**0.90** ±0.19	**0.90** ±0.19	**0.82** ±0.16

Table A.9: Performance as in equation 3 of algorithms averaged over dataset creation seeds and offline runs, where ± captures the standard deviation. Results are for Classic Control and MiniGrid environments on all nine algorithms.

Dataset	BC	DQN	BCQ	CQL
`Breakout-MinAtar-v0`				
Random	0.02 ± 0.00	0.17 ± 0.02	0.16 ± 0.01	**0.33** ± 0.02
Mixed	0.11 ± 0.01	0.13 ± 0.03	**0.57** ± 0.23	0.30 ± 0.03
Replay	0.67 ± 0.05	0.39 ± 0.11	**0.98** ± 0.03	0.88 ± 0.15
Noisy	0.43 ± 0.04	0.22 ± 0.04	**0.83** ± 0.09	**0.82** ± 0.16
Expert	**0.82** ± 0.09	0.01 ± 0.01	**0.81** ± 0.09	**0.77** ± 0.15
`SpaceInvaders-MinAtar-v0`				
Random	0.05 ± 0.01	**1.18** ± 0.10	1.12 ± 0.09	**1.24** ± 0.11
Mixed	0.08 ± 0.03	**0.82** ± 0.35	**0.69** ± 0.17	**0.87** ± 0.20
Replay	0.62 ± 0.02	**1.29** ± 0.07	**1.25** ± 0.07	**1.26** ± 0.07
Noisy	0.37 ± 0.16	**1.16** ± 0.19	0.62 ± 0.33	0.96 ± 0.17
Expert	**0.48** ± 0.26	0.02 ± 0.03	**0.48** ± 0.27	**0.49** ± 0.27

Table A.10: Performance as in equation 3 of algorithms averaged over dataset creation seeds and offline runs, where ± captures the standard deviation. Results are for MinAtar environments on a selection of four algorithms.

A.8 Illustration of State-Action Coverage on MountainCar

In Fig. A.1 we illustrate SACo on the example of the `MountainCar-v0` environment. This environment was chosen as the state-space is two-dimensional and thus provides axes with physical meaning.

In this example, the dataset obtained through a random policy has only limited coverage of the whole state-action space. This is the case, because the random policy is not able to transition far from the starting position due to the environment dynamics.

Furthermore, the expert policies obtained in each independent run differ from one another in how they steer the agent towards the goal, for instance, neglecting to use the action *"Don't accelerate"* in the first run.

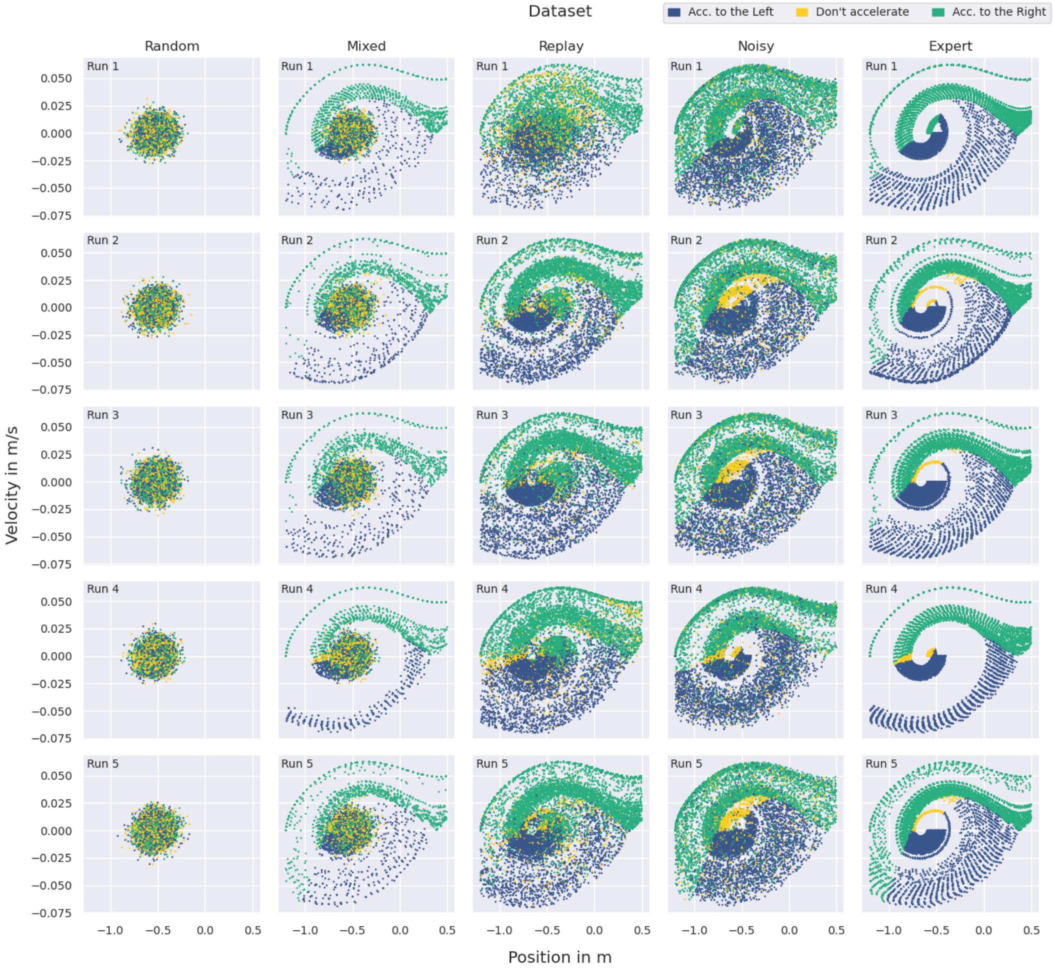

Figure A.1: State-action space for different datasets created from the environment `MountainCar-v0` under different dataset schemes for five independent runs. 10% of the datasets were sub-sampled for plotting.

A.8.1 Performance per Dataset Generation Scheme

To obtain results per dataset generation scheme, the results for the five dataset creation runs per scheme are averaged. Therefore, the relative TQ and SACo are averaged as well as the performance for the respective algorithm on each dataset. Results are depicted in Fig. A.2.

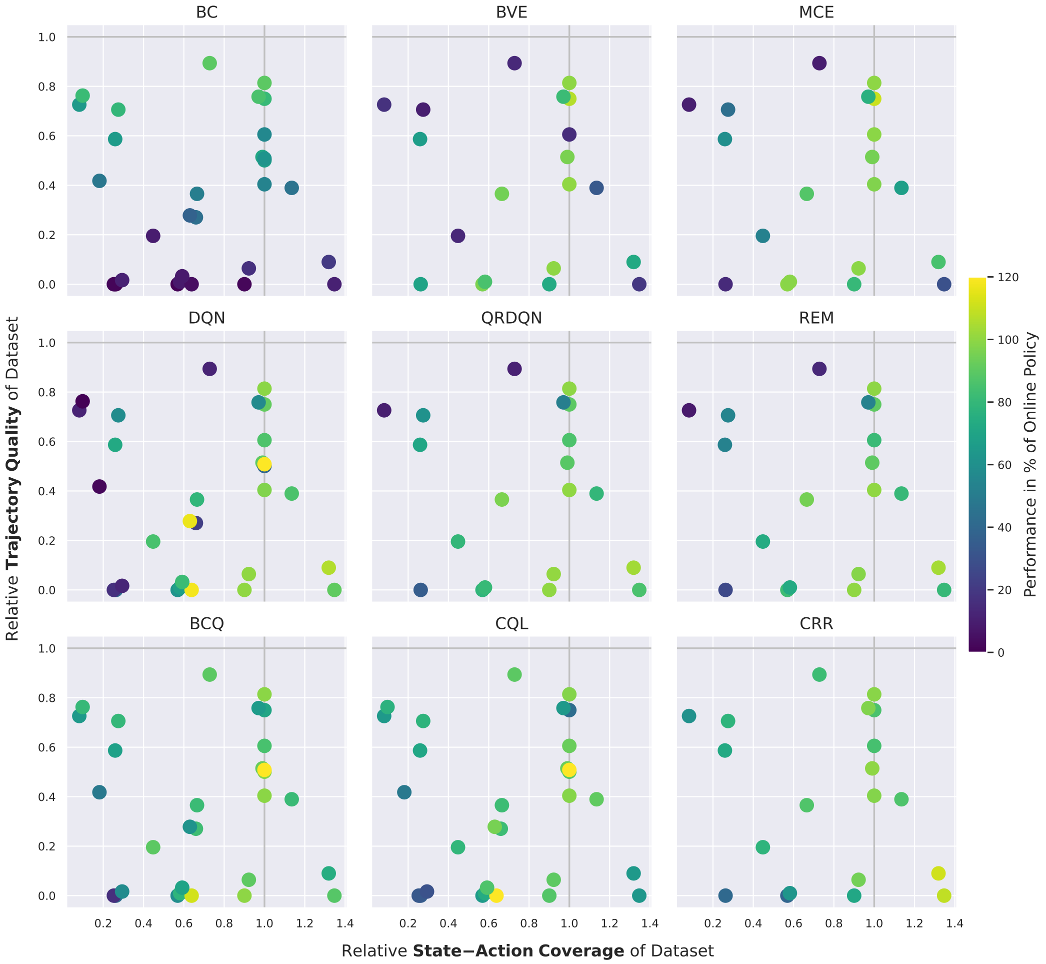

Figure A.2: Performance of algorithms compared to the online policy used to create the datasets, with respect to the relative TQ and SACo of the dataset. Points denote the different datasets, where BC, DQN, BCQ and CQL additionally include results on MinAtar environments. Relative TQ, SACo and performance are averaged over results for each of the five dataset creation seeds.