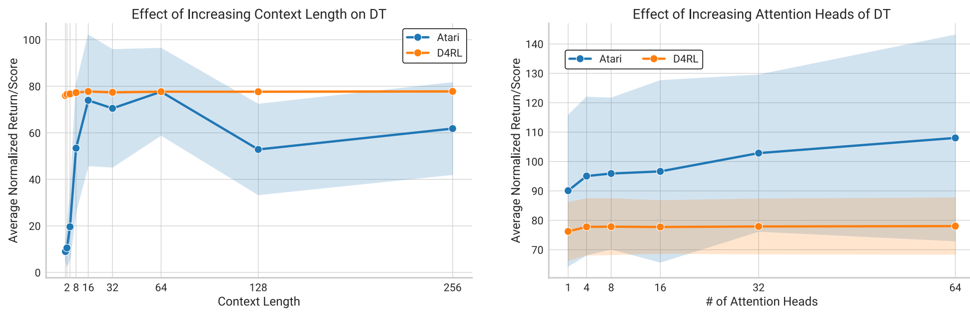


Figure 6: Increasing the context length (left) and number of attention heads (right) of DT on both ATARI and D4RL. ATARI curves average over BREAKOUT, QBERT, SEAQUEST, and PONG. D4RL curves average over all tasks and data splits. We see that scaling context length and attention heads benefits performance on ATARI, but not on D4RL, though the observed benefit of increasing the number of attention heads is not statistically significant.

A Architectural Properties of Decision Transformers

Here, we study the impact of architectural properties of DT, namely the context length, # of attention heads, # of layers, and embedding size. Full experimental results are in Appendix H.

Context length: The use of a context window makes DT dependent on the history of states, actions, and rewards, unlike CQL and BC. Figure 6 (left) demonstrates the role of context length for DT. Having a context length larger than 1 (i.e., no history) did not benefit DT on D4RL, while it helped on ATARI, where performance is maximized at a context length of 64. This finding indicates that some tasks may benefit from a more extensive knowledge of history than others. The deterioration in performance, as context length increases, is likely due to DT overfitting to certain trajectories.

Attention Heads: While the importance of the number of Transformer attention heads has been noted in NLP (Michel et al., 2019), it is an open question how the trends carry over to offline RL. Figure 6 (right) shows the impact of this hyperparameter on DT. We observed monotonic improvement on ATARI, but no improvement on D4RL. One of the primary reasons for this discrepancy is that there is more room for an agent to extract higher rewards on ATARI than D4RL. For instance, DT achieved an *expert-normalized* score of over 320 on the BREAKOUT ATARI game, but never gets over 120 in D4RL. This suggests there is more room for scaling the data/parameters to improve results on ATARI.

Number of layers: This was discussed in Section 4.7, with results in Figure 4.

Embedding size: Increasing the embedding size of DT beyond 256 did not result in any improvement in ATARI; see Table 20 in Appendix H for results.

B Extended Background

In the reinforcement learning (RL) problem, an agent interacts with a Markov decision process (MDP) (Puterman, 1990), defined as a tuple $(\mathcal{S}, \mathcal{A}, T, R, H)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, $T(s', a, s) = \Pr(s'|s, a)$ is the transition model, $R(s, a) \in \mathbb{R}$ is the reward function, and H is the (finite) horizon. At each timestep, the agent takes an action $a \in \mathcal{A}$, the environment state transitions according to T, and the agent receives a reward according to R. The agent does not know T or R. Its objective is to obtain a policy $\pi : \mathcal{S} \to \mathcal{A}$ such that acting under the policy maximizes its return, the sum of expected rewards: $\mathbb{E}[\sum_{t=0}^{H} R(s_t, \pi(s_t))]$. In offline RL (Levine et al., 2020), agents cannot interact with the MDP but instead learn from a fixed dataset of transitions $\mathcal{D} = \{(s, a, r, s')\}$, generated from an unknown behavior policy.

Q-Learning and CQL. One of the most widely studied learning paradigms is Q-Learning (Sutton and Barto, 2018), which uses temporal difference (TD) updates to estimate the value of taking actions from states via bootstrapping. Although Q-Learning promises to provide a general-purpose decision-making framework, in the offline setting algorithms such as Conservative Q-Learning (CQL) (Kumar et al., 2020) and Implicit Q-Learning (IQL) (Kostrikov et al., 2021) are often unstable and highly sensitive to the choice of hyperparameters (Brandfonbrener et al., 2022). In this paper, we will focus on Conservative Q-Learning (CQL). CQL (Kumar et al., 2020) proposes a modification to the standard Q-learning algorithm to address the overestimation problem by constraining the Q-values so that they do not exceed a lower bound. This constraint is highly effective in the offline setting because it mitigates the distributional shift between the behavior policy and the learned policy.

More concretely, CQL adds a regularization term to the standard Bellman update:

$$\min_{\theta}\ \alpha\left(\mathbb{E}_{s\sim\mathcal{D}}\left[\log\left(\sum_{a'}\exp(Q_\theta(s,a'))\right)\right]-\mathbb{E}_{s,a\sim\mathcal{D}}\left[Q_\theta(s,a)\right]\right)+\mathsf{TDError}(\theta;\mathcal{D}), \tag{1}$$

where α is the weight given to the first term, and the second term is the distributional $\mathsf{TDError}(\theta;\mathcal{D})$ under the dataset, from C51 (Bellemare et al., 2017).

Imitation Learning and BC. When the dataset comes from expert demonstrations or is otherwise near-optimal, researchers often turn to imitation learning algorithms such as Behavior Cloning (BC) or TD3+BC (Fujimoto and Gu, 2021b) to train a policy. BC is a simple imitation learning algorithm that performs supervised learning to map states s in the dataset to their associated actions a. Due to the reward-agnostic nature of BC, it typically requires expert demonstrations to learn an effective policy. The clear downside of BC is that the imitator cannot be expected to attain higher performance than was attained in the dataset.

Sequence Modeling and DT. Sequence modeling is a recently popularized class of offline RL algorithms that includes the Decision Transformer (Chen et al., 2021) and the Trajectory Transformer (Janner et al., 2021). These algorithms train autoregressive sequence models that map the history of the trajectory to the next action. In the Decision Transformer (DT) model, the learned policy produces action distributions conditioned on trajectory history and desired returns-to-go $\hat{R}_{t'}=\sum_{t=t'}^{T} r_t$. This leads to the following trajectory representation, used for both training and inference: $\tau=(\hat{R}_1,s_1,a_1,\hat{R}_2,s_2,a_2,\dots,\hat{R}_T,s_T,a_T)$. Conditioning on returns-to-go enables DT to learn effective policies from suboptimal data and produce a range of behaviors during inference.

C Additional Dataset Details

Humanoid Data In this section, we present the details of the HUMANOID offline Reinforcement Learning (RL) dataset created for our experiments. We trained a Soft Actor-Critic (SAC) agent (Haarnoja et al., 2018) for 3 million steps and selected the best-performing agent, which achieved a score of 5.5k, to generate the expert split. To create the medium split, we utilized an agent that performed at one-third of the expert's performance. We then generated the medium-expert split by concatenating the medium and expert splits. Our implementation is based on (Raffin et al., 2021) and employs the default hyperparameters for the SAC agent (behavior policy). Table 5 displays the performance of all agents across all splits of the HUMANOID task. Furthermore, Table 6 provides statistical information on the HUMANOID dataset.

Task	BC	DT	CQL
Humanoid Medium	13.22 $\pm$ 4.25	23.67 $\pm$ 2.48	**49.51 $\pm$ 0.77**
Humanoid Medium Expert	13.67 $\pm$ 5.21	52.63 $\pm$ 1.92	**54.1 $\pm$ 1.36**
Humanoid Expert	24.22 $\pm$ 5.37	53.41 $\pm$ 1.32	**63.69 $\pm$ 0.22**
Average	17.03	43.23	55.76

Table 5: Performance of agents on a high dimensional state space task. CQL seems better suited to tasks involving higher state space dimensions.

Task	# of trajectories	# of samples
Humanoid Medium	4000	1.76M
Humanoid Expert	4000	3.72M
Humanoid Medium Expert	8000	5.48M

Table 6: Dataset statistics for HUMANOID. The number of samples refers to the number of environment transitions recorded in the dataset.

Robomimic: We visualize the return distribution of ROBOMIMIC tasks, employing a discount factor of 0.99, as illustrated in Figure 7 and Figure 8. It becomes evident that the discount factor has a significant impact on the data's optimality characteristics. PH features shorter trajectories, resulting in a higher proportion of high-return data.

D Additional Evaluation Details

We specify our sampling procedure used to evaluate the ATARI benchmark, used in Section 4.7 and Section A. The **ATARI** offline dataset (Agarwal et al., 2020) contains the interaction of a DQN agent (Mnih et al., 2015) as

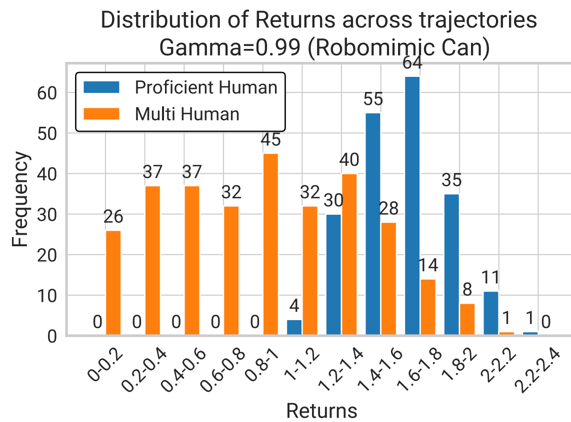


Figure 7: Return Distribution in Proficient and Multi Human splits of Robomimic Can environment shown with gamma set to 0.99. Proficient Human trajectories are shorter in length compared to Multi Human trajectories. Even though PH and MH datasets contain the same number of trajectories, the PH dataset contains more near-expert data.

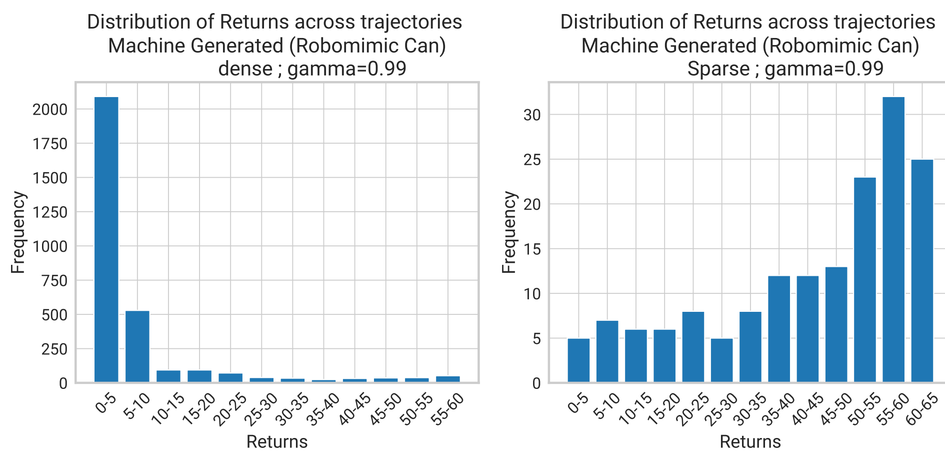


Figure 8: Reward Distribution in Machine Generated splits of Robomimic Can environment is shown with gamma set to 0.99. The sparse variant has more high return trajectories.

it is trained progressively in 50 buffers. Each observation in the dataset contains the last 4 frames of the game, stacked together. Buffers 1 and 50 would contain interactions of the DQN agent when it is naive and expert respectively. Our results are averaged across four different experiments. Each experiment performed a sampling of 500k timestep data from the buffers numbered 1) 1-50 2) 40-50 (DQN is competitive), 3) 45-50, and 4) 49-50 (DQN is expert). We study the architectural and scaling properties of DT in this dataset, where we consider four games: BREAKOUT, QBERT, SEAQUEST, and PONG. We follow the protocol of Lee et al. (2022) and Kumar et al. (2023) by training on the Atari DQN Replay dataset, which used sticky actions, but then evaluating with sticky actions disabled.

E Disentangling DT and BC

In this section, we experimented with an additional baseline, "BC Transformer," a modified version of DT that does not perform conditioning on a returns-to-go vector and has a context length of 1. As previously mentioned, we set the context length to 1 in the case of DT for running experiments on ROBOMIMIC as well. This section aims to investigate the discrepancy between the performance of DT and BC, specifically, we wanted to understand how much of the discrepancy can be attributed to RTG conditioning compared to architectural differences between DT and BC which is typically implemented using a stack of MLPs. By having the BC Transformer baseline, the only distinguishing component between it and BC is the architecture. We observed that BC is often a better-performing agent than BC Transformer on PH Table 7, MG Table 8, and MH tasks Table 9.

Additionally, it can also be observed that RTG conditioning only plays a critical role when the distribution of reward shows variation. Unlike expert data such as PH and MH where the RTG vector remained the same, we see DT outperforming BC Transformer on MG significantly.

Layers	**BC Transformer**	**DT**	**BC**	**CQL**
Lift-PH	100 ± 0	100 ± 0	100 ± 0	92.7 ± 5
Can-PH	94.6 ± 2.4	96 ± 0	95.3 ± 0.9	38 ± 7.5
Square-PH	52 ± 5.8	53.3 ± 2.4	78.7 ± 1.9	5.3 ± 2.5
Transport-PH	0 ± 0	1.3 ± 0	29.3 ± 0.9	0 ± 0

Table 7: Results of agents on ROBOMIMIC PH tasks

Layers	**BC Transformer**	**DT**	**BC**	**CQL**
Lift-MG-sparse	77.9 ± 7.11	96 ± 2.1	65.3 ± 2.5	64 ± 2.8
Can-MG-sparse	48.66 ± 1.8	92.8 ± 4.1	64.7 ± 3.4	1.3 ± 0.9
Lift-MG-dense	68 ± 3.2	93.99 ± 2.5	60 ± 2	63.3 ± 5.2
Can-MG-dense	43.99 ± 3.2	82.4 ± 4.2	64 ± 4.3	0 ± 0

Table 8: Results of agents on ROBOMIMIC MG tasks

Layers	**BC Transformer**	**DT**	**BC**	**CQL**
Lift-MH	100 ± 0	100 ± 0	100 ± 0	56.7 ± 40.3
Can-MH	93.3 ± 4.1	95.33 ± 0	86 ± 4.3	22 ± 5.7
Square-MH	18.66 ± 4.1	21.33 ± 3.7	52.7 ± 6.6	0.7 ± 0.9
Transport-MH	0 ± 0	0 ± 0	11.3 ± 2.5	0 ± 0

Table 9: Results of agents on ROBOMIMIC MH tasks

F ADDITIONAL RESULTS ON D4RL AND ROBOMIMIC

This section contains results obtained on individual tasks of D4RL and ROBOMIMIC benchmarks. We use the averaged-out results obtained on all of the tasks from the respective benchmark for our analysis.

F.1 ESTABLISHING BASELINES

This section presents baseline results for individual tasks in both sparse and dense settings of the D4RL. The average outcomes are detailed in Table 2 (Section subsection 4.1). Our observations indicate that DT consistently outperforms CQL and BC in nearly all tasks within the sparse setting of the D4RL benchmark. Although CQL achieves a marginally higher average return on the Hopper task (3.4% ahead of DT for medium and medium-replay splits), it also exhibits significantly higher volatility, as evidenced by the standard deviations. In contrast, DT remains competitive and robust. In the sparse reward setting, CQL surpasses BC by 5.2%. As highlighted in Section subsection 4.1, CQL is most effective in the dense reward setting of the D4RL benchmark.

Dataset	DT		CQL		BC
	Sparse	Dense	Sparse	Dense	
Medium					
Half Cheetah	**42.54** $\pm$ **0.12**	42.55 $\pm$ 0.02	38.63 $\pm$ 0.81	47.03 $\pm$ 0.075	42.76 $\pm$ 0.17
Hopper	69.75 $\pm$ 2.31	73.03 $\pm$ 0.74	**73.89** $\pm$ **10.12**	70.54 $\pm$ 0.41	64.35 $\pm$ 5.6
Walker	**75.42** $\pm$ **1.07**	75.42 $\pm$ 0.9	19.31 $\pm$ 3.17	83.77 $\pm$ 0.25	54.62 $\pm$ 12.04
Average	62.56 $\pm$ 1.16	63.66 $\pm$ 0.55	43.94 $\pm$ 4.7	67.11 $\pm$ 0.24	53.91 $\pm$ 5.93
Medium Replay					
Half Cheetah	**38.76** $\pm$ **0.26**	37.09 $\pm$ 0.4	35 $\pm$ 2.56	46.12 $\pm$ 0.06	9.81 $\pm$ 9.2
Hopper	82.24 $\pm$ 1.58	89.15 $\pm$ 1.19	**83.1** $\pm$ **19.21**	101.27 $\pm$ 0.46	16.19 $\pm$ 10.8
Walker	**71.24** $\pm$ **1.93**	69.44 $\pm$ 3.14	29.02 $\pm$ 19.63	87.85 $\pm$ 0.83	17.82 $\pm$ 4.96
Average	64.08 $\pm$ 1.25	65.22 $\pm$ 1.57	49.04 $\pm$ 13.79	78.41 $\pm$ 0.45	14.6 $\pm$ 8.32
Medium Expert					
Half Cheetah	**90.55** $\pm$ **1.53**	91.67 $\pm$ 0.21	24.35 $\pm$ 2.38	94.95 $\pm$ 0.25	42.95 $\pm$ 0.14
Hopper	**110.29** $\pm$ **0.58**	110.76 $\pm$ 0.06	42.44 $\pm$ 12.52	109.67 $\pm$ 2.12	62.21 $\pm$ 6.5
Walker	**108.63** $\pm$ **0.22**	108.49 $\pm$ 0.1	21.3 $\pm$ 0.55	111.58 $\pm$ 0.17	38 $\pm$ 9.86
Average	103.15 $\pm$ 0.77	103.64 $\pm$ 0.12	29.36 $\pm$ 5.14	105.39 $\pm$ 0.84	47.72 $\pm$ 5.5
Total Average	**76.6** $\pm$ **1**	77.51 $\pm$ 1.12	40.78 $\pm$ 7.88	**83.64** $\pm$**0.51**	38.74 $\pm$ 6.58

Table 10: Results on D4RL in both sparse and dense reward settings, averaged over the HALFCHEETAH, HOPPER, and WALKER tasks. The presented results here are an extension of Table 2.

F.2 How does the amount and quality of data affect each agent's performance?

Figure 9 illustrates the performance of agents on individual tasks within the D4RL benchmark as data quality and quantity are varied. DT improved or plateaued (upon reaching maximum performance) as additional data was provided. In contrast, CQL exhibits volatility, displaying significant performance declines in HOPPER and WALKER2D medium-replay tasks. The performance of BC tends to deteriorate when trained on low-return data.

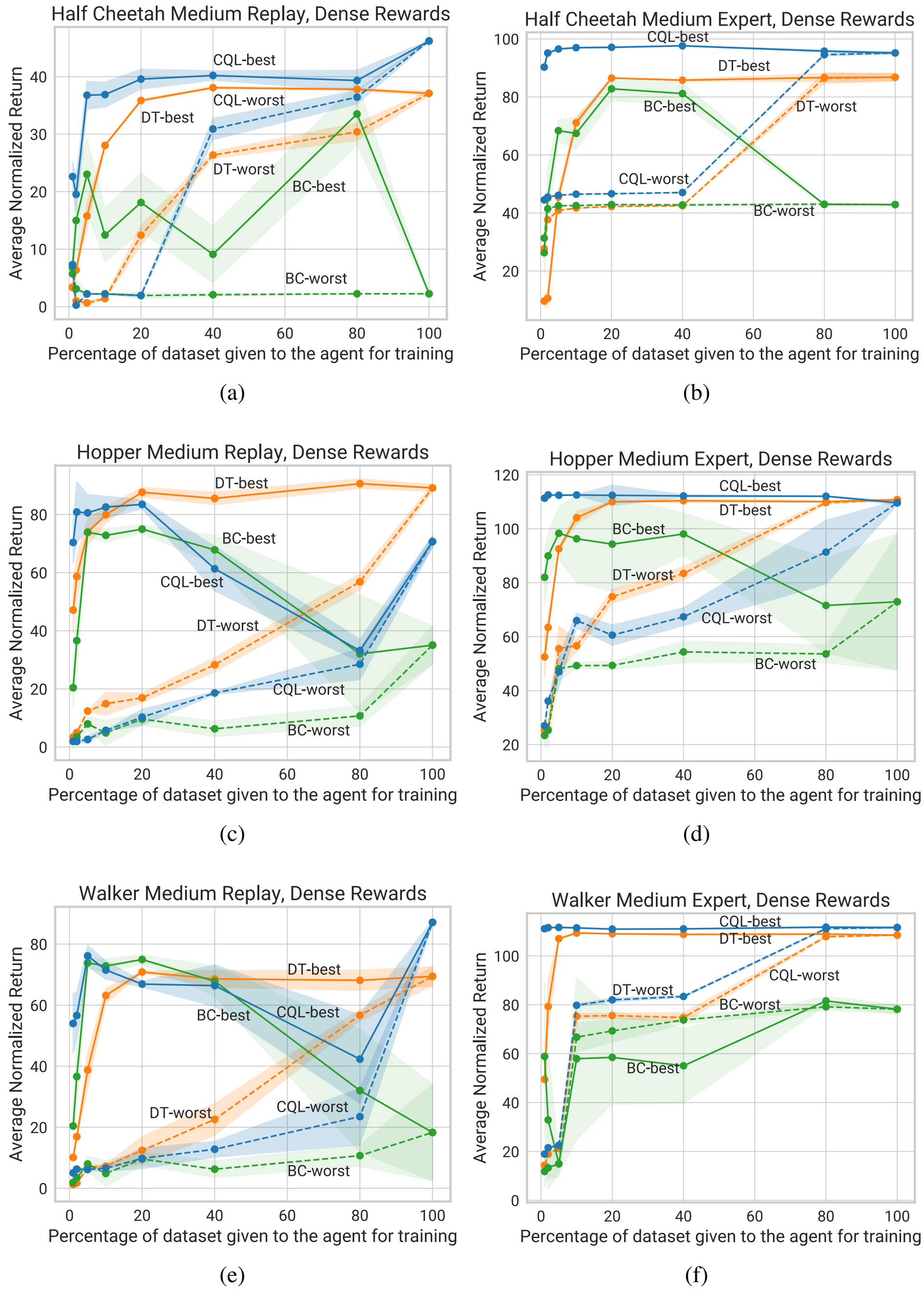
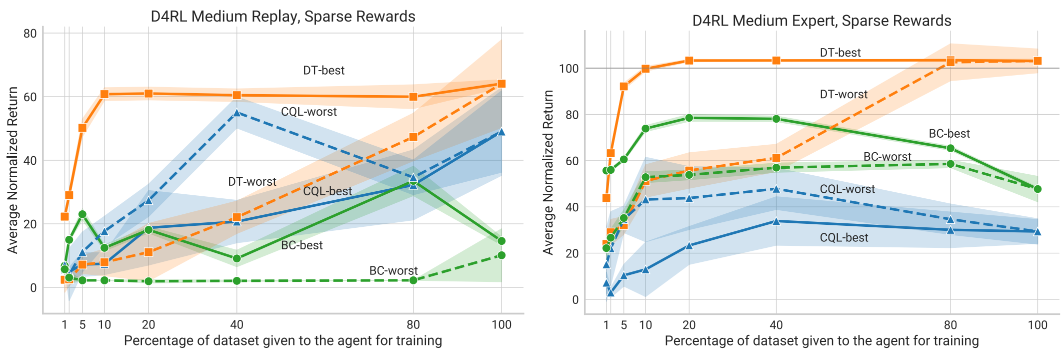


Figure 9: Results demonstrating the behavior of agents as the amount of data and quality is varied on medium-replay and medium-expert splits of D4RL benchmark in dense reward setting. The results presented here are an extension of Figure 1.

Figure 10 illustrates the performance behavior of agents as the quantity and quality of data are adjusted in a sparse setting on the D4RL dataset. A more detailed exploration of this behavior across individual tasks is presented in Figure Figure 11.

Two key observations can be made from these results. 1) In the sparse reward setting, DT becomes a markedly more sample-efficient choice, with its performance either improving or remaining steady as the quantity of data increases. In contrast, CQL displays greater variability and fails to exceed BC in scenarios involving expert data (medium-expert). 2) The sub-optimal data plays a significantly more important role for CQL in sparse settings as compared to dense settings. Our hypothesis is that the sparsity of feedback makes learning about course correction more critical than learning from expert demonstrations. Notably, we discovered that the worst 10% of data features trajectories with substantially higher return coverage, which contributes to greater diversity within the data. This, in turn, enhances CQL's ability to learn superior Q-values (course correction) when compared to the best 10% of data in a medium-expert data setting.



(a) (b)

Figure 10: Normalized D4RL returns obtained by training DT, CQL, and BC on various amounts of highest-return ("best") or lowest-return ("worst") data in sparse reward setting. The left plot (a) is on medium replay data, while the right plot (b) is on medium expert data; both plots average over the HALFCHEETAH, HOPPER and WALKER tasks. Notice that the Y-axis limits are set lower on the left plot. We observed that DT was a substantially more sample-efficient agent than CQL. Sub-optimal data was noticeably crucial for CQL in sparse reward setting.

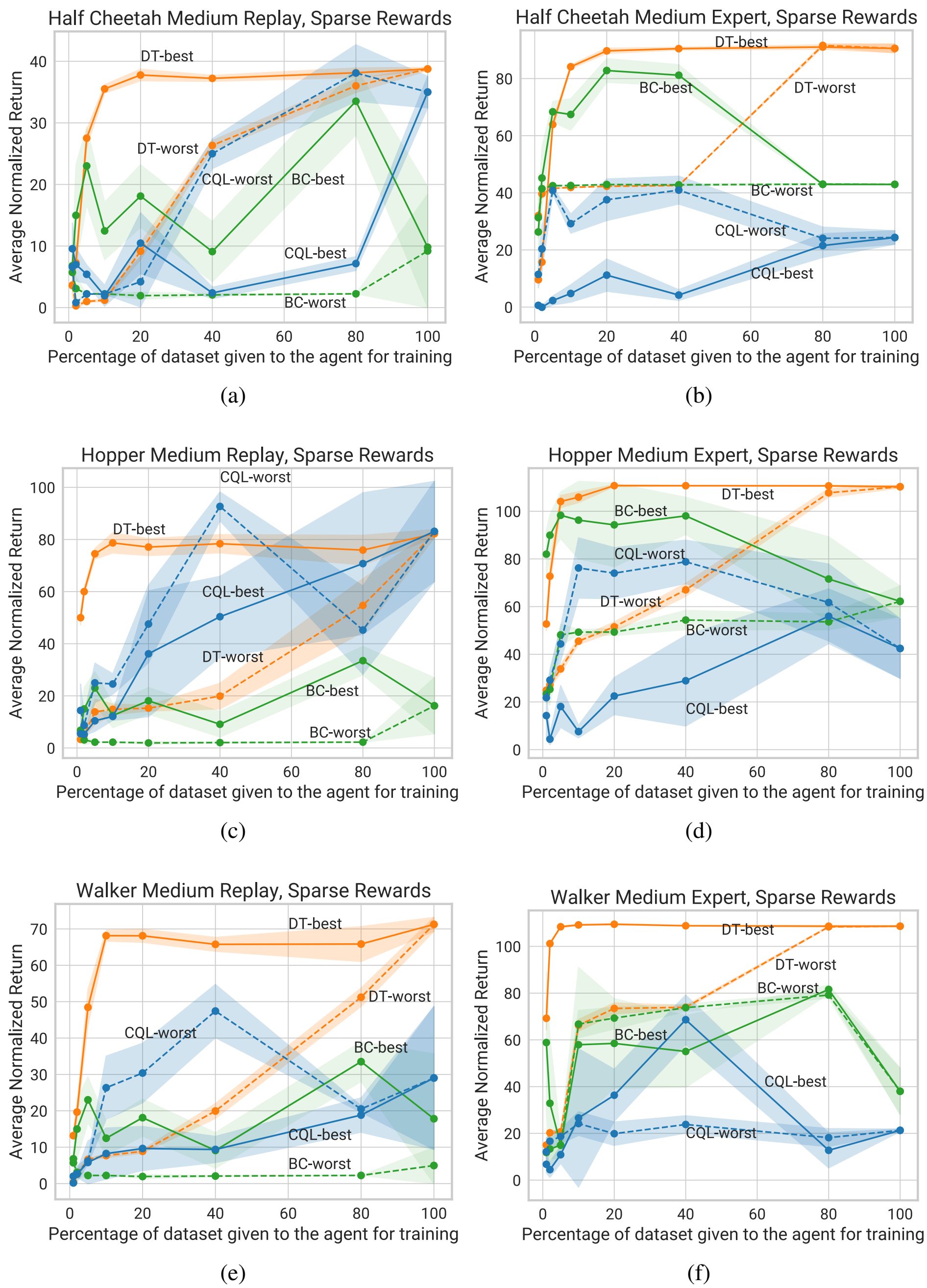


Figure 11: Results demonstrating the behavior of agents as the amount of data and quality is varied on medium-replay and medium-expert splits of D4RL benchmark in sparse reward setting.

F.3 How are agents affected when trajectory lengths in the dataset increase?

Table 11 showcases the performance of all agents across individual ROBOMIMIC tasks, encompassing both synthetic and human-generated datasets. DT surpasses both agents in all synthetic tasks within the ROBOMIMIC benchmark, for both sparse and dense settings. Interestingly, BC demonstrates a strong aptitude for numerous tasks with human-generated data, particularly excelling on the SQUARE task.

Dataset Type	Average Trajectory Length	DT	CQL	BC
Lift MG Dense	150 ± 0	**96 ± 1.2**	68.4 ± 6.2	59.2±6.19
Lift MG Sparse	150 ± 0	**93.2 ± 3.2**	60±13.2	59.2±6.19
Can MG Dense	150 ± 0	**83.2 ± 0**	2 ± 1.2	49.6±2.4
Can MG Sparse	150 ± 0	**83.2 ± 1.59**	0 ± 0	49.6±2.4
Lift-PH	48 ± 6	**100 ± 0**	92.7 ± 5	**100**±0
Lift-MH-Better	72 ± 24	94.6 ± 1.8	88 ± 5.9	**98.7 ± 1.9**
Lift-MH-Okay-Better	83 ± 29	100 ± 0	86 ± 6.5	**99.3 ± 0.9**
Lift-MH-Okay	94 ± 30	**97.33 ± 0**	67.3 ± 10.5	**96 ± 1.6**
Lift-MH	104 ± 44	**100 ± 0**	56.7 ± 5	**100 ± 0**
Lift-MH-Worse-Better	109 ± 49	**100 ± 0**	75.3 ± 25.6	**100 ± 0**
Can-PH	116 ± 14	**96 ± 0**	38.7 ± 7.5	**95.3**±0.9
Lift-MH-Worse-Okay	119 ± 44	**100 ± 0**	64.7 ± 2.5	**98.7 ± 1.9**
Can-MH-Better	143 ± 29	52.66 ± 0	20.7 ± 7.4	**83.3 ± 2.5**
Lift-MH-Worse	145 ± 40	94.6 ± 0	13.3 ± 9	**100 ± 0**
Square-PH	151 ± 20	53.3 ± 2.4	5.3 ± 2.5	**78.7 ± 1.9**
Can-MH-Okay-Better	162 ± 44	75.33 ± 3.3	30.7 ± 7.7	**90.7 ± 1.9**
Can-MH-Okay	181 ± 47	52.66 ± 0	22 ± 4.3	**72 ± 2.8**
Square-MH-Better	185 ± 46	13.3 ± 0	0.7 ± 0.9	**58.7 ± 2.5**
Can-MH	209 ± 114	**95.33 ± 0**	22 ± 7.5	86 ± 4.3
Can-MH-Worse-Better	224 ± 134	**73.99 ± 1.1**	20.7 ± 5.7	**76 ± 4.3**
Square-MH-Okay-Better	225 ± 76	20.66 ± 0	1.3 ± 0.9	**56.7 ± 4.1**
Can-MH-Worse-Okay	242 ± 126	54 ± 1.6	18.8 ± 2.5	**74.7 ± 5.7**
Square-MH-Okay	265 ± 78	12.6 ± 0	0 ± 0	**27.3 ± 3.4**
Square-MH	269 ± 123	21.33 ± 3.7	0.7 ± 0.9	**52.7 ± 6.6**
Square-MH-Worse-Better	271 ± 140	6 ± 0	1.3 ± 0.9	**46.7 ± 5.7**
Can-MH-Worse	304 ± 148	51.33 ± 0	4 ± 3.3	**56.7 ± 2.5**
Square-MH-Worse-Okay	311 ± 128	4.6 ± 0	2.7 ± 1.9	**28.7 ± 2.5**
Square-MH-Worse	357 ± 150	11.3 ± 0	0 ± 0	**22 ± 4.3**

Table 11: Success rates for different synthetic and human-generated datasets in ROBOMIMIC. Results presented here are an extension of Table 4 (from subsection 4.3) and have been sorted as per the trajectory length of the task. DT outperformed both agents on synthetic data as seen by performance on MG tasks. BC seemed well suited to numerous tasks for which data was human-generated (PH and MH). Bold numbers represent the best-performing agent on the corresponding task.

F.4 How are agents affected when suboptimal data is added to the dataset?

Figure 12 depicts the behavior of agents when random data is introduced according to “Strategy 1” in the dense reward data regime of the D4RL benchmark. As previously described, “Strategy 1” involves rolling out a uniformly random policy from sampled initial states to generate random data. Our observations indicate that both CQL and DT maintain stable performance, while BC exhibits instabilities, occasionally failing as observed in the HALF CHEETAH task.

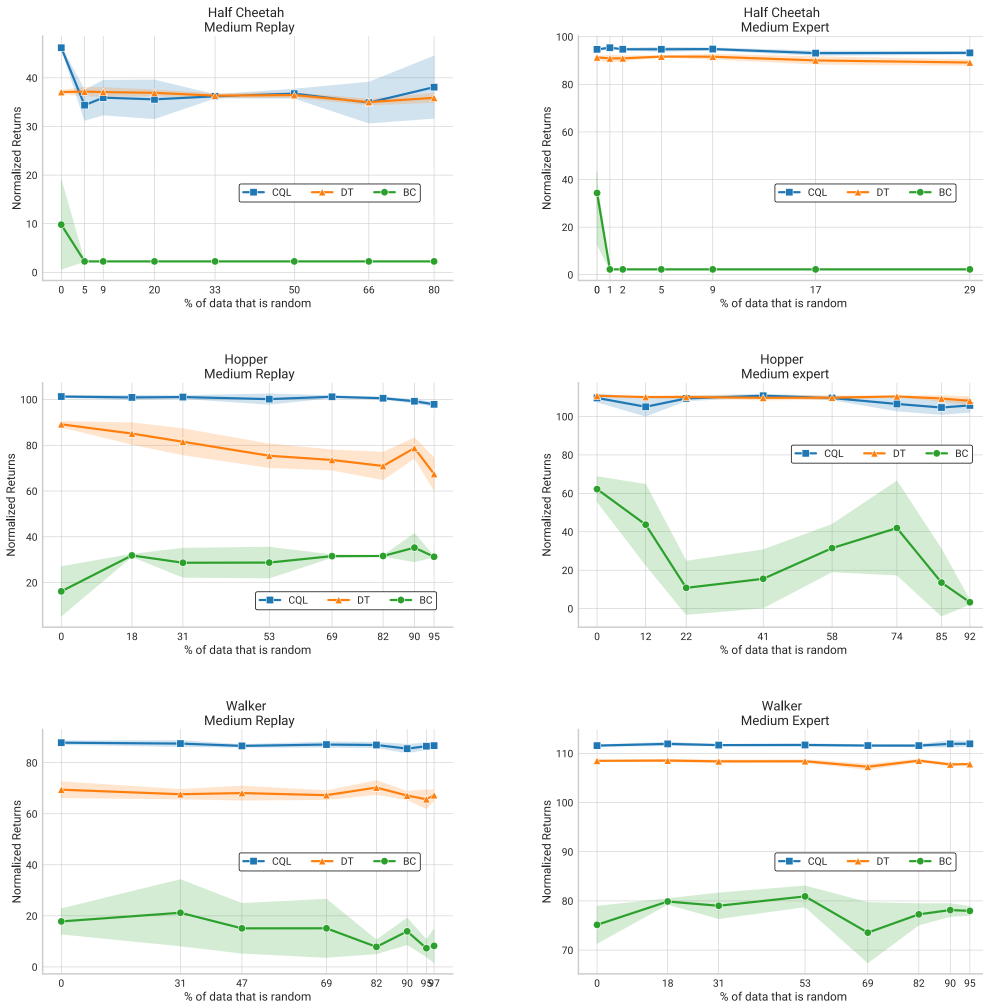


Figure 12: Impact of adding suboptimal data on all agents in dense reward data regime on D4RL tasks using Strategy 1. The results presented here are an extension to Figure 2 (from subsection 4.4).

Figure 13 displays the behavior of agents as random data is incorporated according to Strategy 2 in the dense reward data regime of the D4RL benchmark. In "Strategy 2," we roll out a pre-trained agent for a certain number of steps, execute a single uniformly random action, and then repeat the process. While Strategy 1 produces random transitions primarily clustered around initial states, Strategy 2 generates random transitions across the entire manifold of states, spanning from initial states to high-reward goal states.

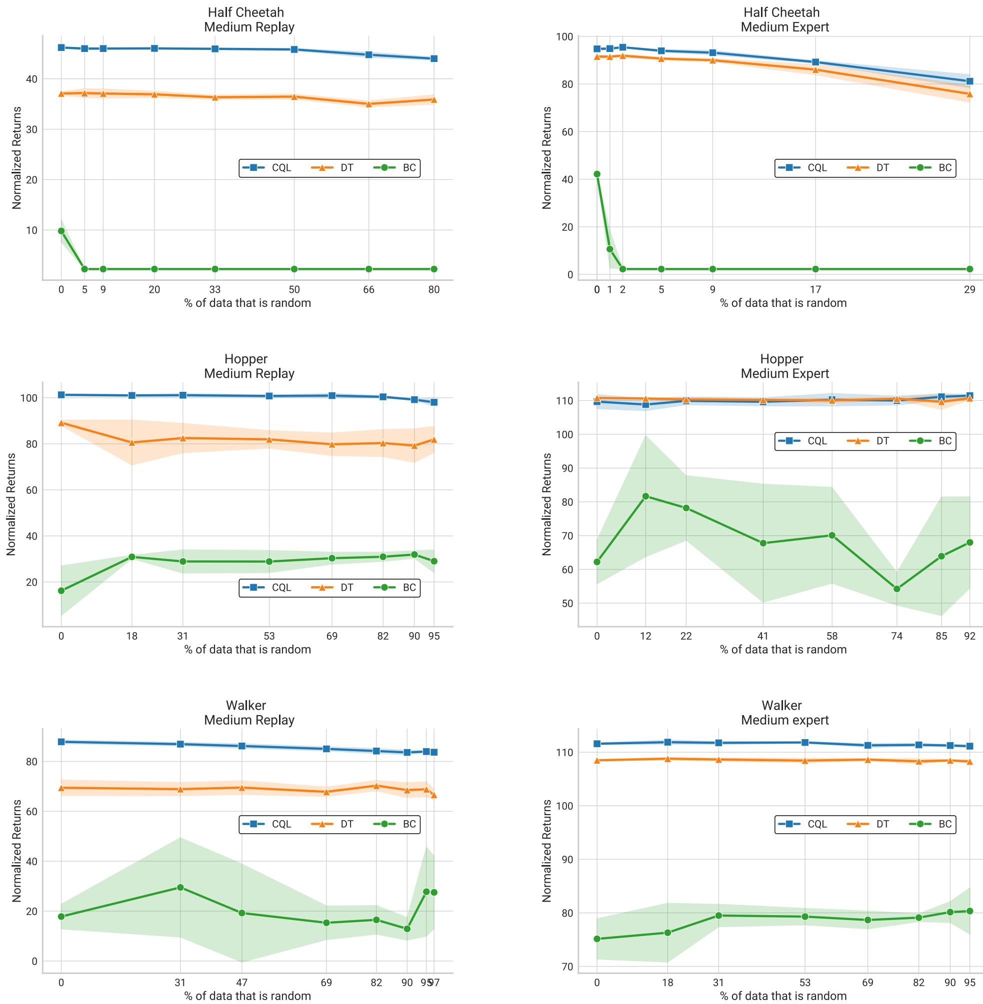


Figure 13: Impact of adding suboptimal data in dense reward data regime on D4RL tasks using Strategy 2. The results presented here are an extension to Figure 2 (from subsection 4.4). A noteworthy observation from this experiment is that the performance of CQL improved on HOPPER medium-expert task with the addition of random data. This suggests that the random data might be aiding in the learning of more accurate Q-values for various states.

Figure 14 illustrates the behavior of agents when random data is introduced according to Strategy 2 in the sparse reward data regime of the D4RL benchmark. We observed that the performance of CQL drastically declined on the HOPPER-MEDIUM-REPLAY task, while its performance stayed the same on other tasks.

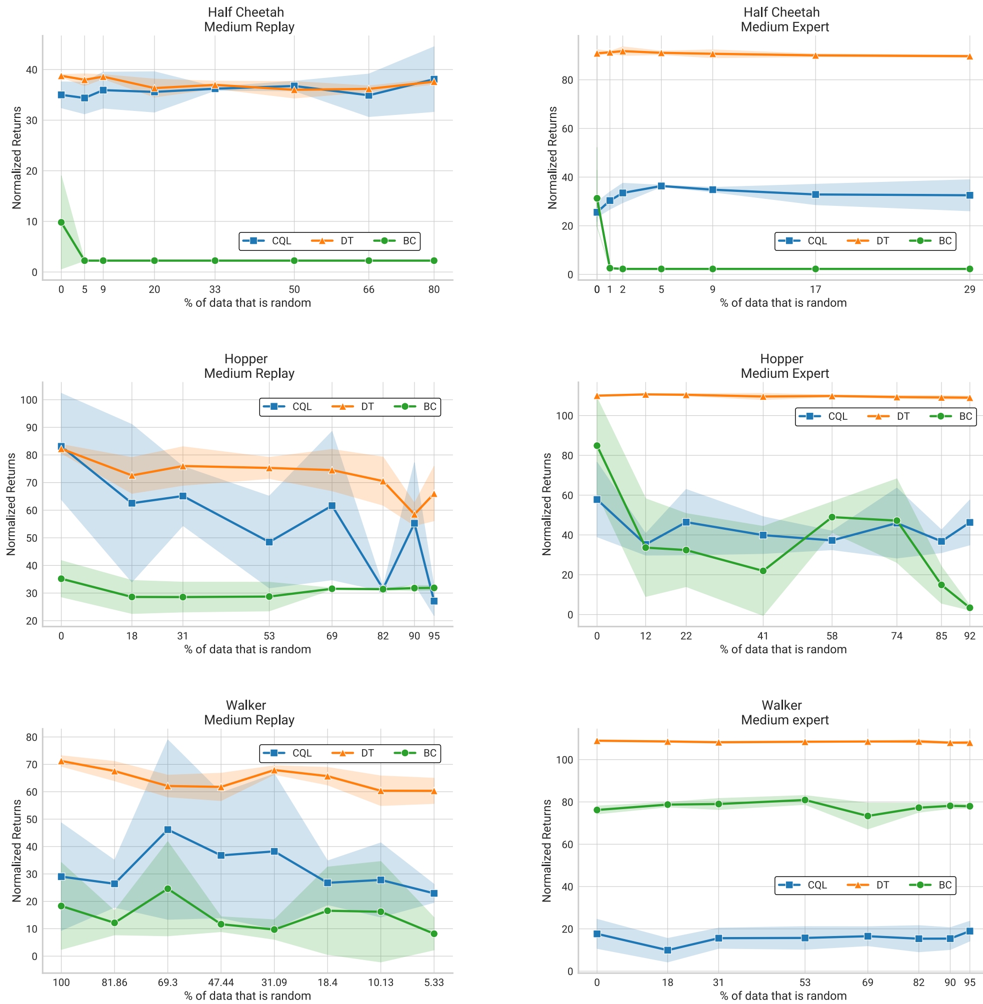


Figure 14: Impact of adding suboptimal data in sparse reward data regime on D4RL tasks using Strategy 2. The results presented here are an extension to Figure 2 (from subsection 4.4). The performance of CQL sharply decreases in the HOPPER medium-replay task. Unlike its performance in the dense reward data regime, CQL does not exhibit the same level of resilience in the sparse reward setting and demonstrates considerably higher volatility.

Figure 15 depicts the behavior of agents when random data is incorporated following Strategy 1 in the sparse reward data regime (human-generated) of the ROBOMIMIC benchmark. Notably, we observed drastically different performance trends with CQL. Its performance plummeted by 80% when the proportion of random data reached 51%. In contrast, its performance nearly doubled on the CAN MG task when the proportion of random data was increased to 30%.

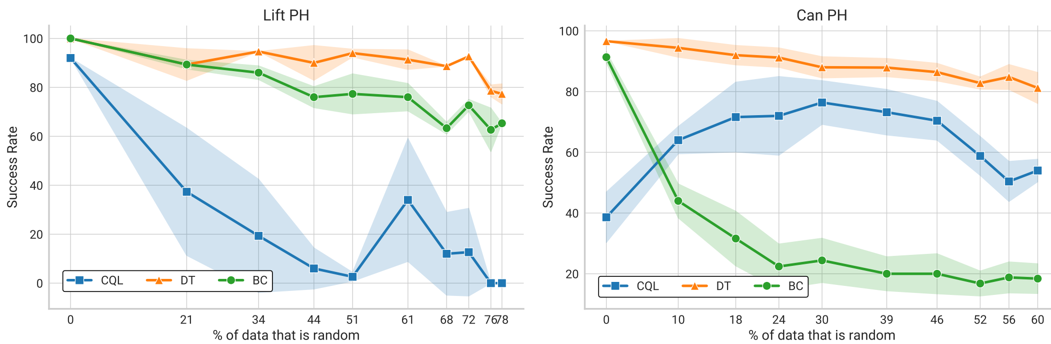


Figure 15: Impact of adding suboptimal data in sparse reward data regime in ROBOMIMIC using Strategy 1. The results presented here are an extension to Figure 2 (from subsection 4.4). CQL exhibited markedly different behaviors across these tasks. Its performance declined sharply on the LIFT PH task. In contrast, its performance significantly improved, nearly doubling, with the addition of random data on CAN PH task.

Figure 16 illustrates the behavior of agents when random data is introduced according to Strategy 2 in the sparse reward data regime (human-generated) of the ROBOMIMIC benchmark.

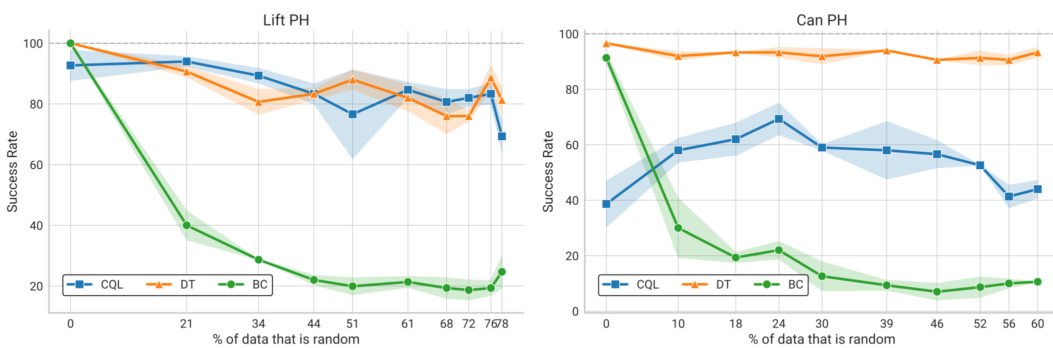


Figure 16: Impact of adding suboptimal data in sparse reward data regime in ROBOMIMIC using Strategy 2. The results presented here are an extension to Figure 2 (from subsection 4.4). Interestingly, when following Strategy 2 for random data generation, CQL maintains stable performance in the LIFT PH task. We observed a similar behavior to that seen with Strategy 1 on the CAN PH task.

Figure 17 and Figure 18 presents the behavior of agents as random data is added following Strategy 2 in sparse and dense reward data regime (synthetic) respectively on the ROBOMIMIC benchmark. In all four scenarios, DT maintained its peak performance reasonably well, indicating its resilience in very noisy data settings. Conservative Q-Learning (CQL), however, showed signs of deterioration on both dense and sparse variants of LIFT tasks. Notably, CQL failed to perform on the CAN task. As mentioned earlier, BC didn't exhibit deterioration, possibly because the Mixed-Goal (MG) data already contains a substantial amount of highly sub-optimal data.

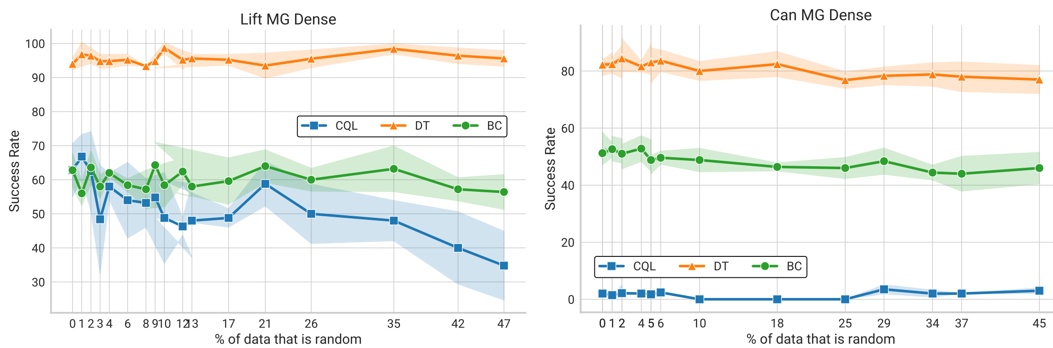


Figure 17: Impact of adding suboptimal data in dense reward data regime in ROBOMIMIC using Strategy 2. The results presented here are an extension to Figure 2 (from subsection 4.4).

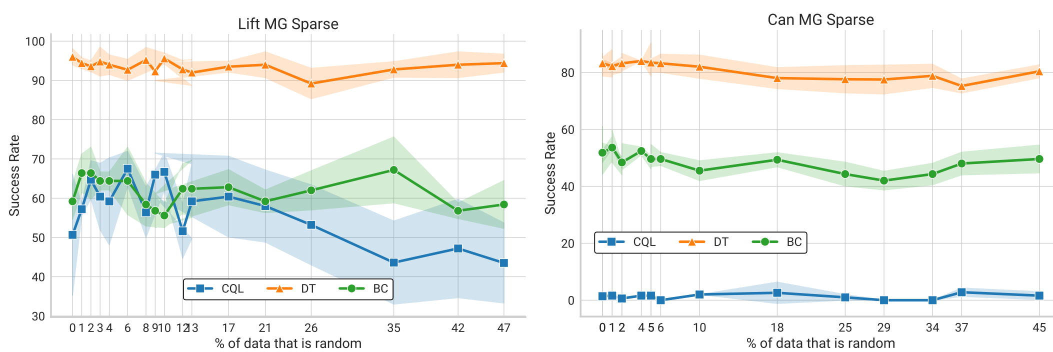


Figure 18: Impact of adding suboptimal data in sparse reward data regime in ROBOMIMIC using Strategy 2. The results presented here are an extension to Figure 2 (from subsection 4.4.

G Additional Experimental Details

We used the original author implementation as a reference for our experiments wherever applicable. In settings where a new implementation was required, we referenced the implementation which has been known to provide competitive/state-of-the-art results on D4RL. We provide details on compute and hyperparameters below.

Compute All experiments were run on an A100 GPU. Most experiments with DT typically require 10-15 hours of training. Experiments with CQL and BC require 5-10 hours of training. We used Pytorch 1.12 for our implementation.

Hyperparameters We mentioned all the hyperparameters used across various algorithms below. Our implementations are based on original author-provided implementations, without any modifications to the hyperparameters. To learn more about the selection of hyperparameters, we recommend viewing the associated papers. Due to the stable training objective of DT and BC, both of these agents do not require substantial hyperparameter sweep experiments. DT was trained using Adam optimizer (Kingma and Ba, 2014) with a Multi-Step Learning Rate Scheduler. Each experiment was run five times to account for seed variance.

Hyperparameter	**RETURNS-TO-GO**
Robomimic PH	6
Robomimic MH	6
Robomimic MG	120

Table 12: Returns-to-go values used with DT on D4RL and ROBOMIMIC.

Hyperparameter	**VALUE**
Reference implementation	https://github.com/kzl/decision-transformer/tree/master/atari (MIT License)
Number of attention heads	8
Number of layers	6
Embedding dimension	128
Context Length (Breakout)	30
Context Length (Qbert)	30
Context Length (Seaquest)	30
Context Length (Pong)	50
Number of buffers	50
Batch size	16
Learning Rate (LR)	$6e-4$
Number of Steps	500000
Return-to-go conditioning	90 Breakout ($\approx 1\times$ max in dataset)
	2500 Qbert ($\approx 5\times$ max in dataset)
	20 Pong ($\approx 1\times$ max in dataset)
	1450 Seaquest ($\approx 5\times$ max in dataset)
Nonlinearity	ReLU, encoder
	GeLU, otherwise
Encoder channels	$32, 64, 64$
Encoder filter sizes	$8\times 8, 4\times 4, 3\times 3$
Encoder strides	$4, 2, 1$
Max epochs	5
Dropout	0.1
Learning rate	$6 * 10^{-4}$
Adam betas	$(0.9, 0.95)$
Grad norm clip	1.0
Weight decay	0.1
Learning rate decay	Linear warmup and cosine decay (see code for details)
Warmup tokens	$512 * 20$
Final tokens	$2 * 500000 * K$

Table 13: Hyperparameters used with DT on ATARI.

Hyperparameter	Value
Reference implementation	https://github.com/kzl/decision-transformer/tree/master/gym (MIT License)
Number of layers	3
Number of attention heads	1
Embedding dimension	128
Nonlinearity function	ReLU
Batch size	512
Return-to-go conditioning (N/A to BC)	6000 HalfCheetah 3600 Hopper 5000 Walker 50 Reacher 6000 Humanoid
Dropout	0.1
Learning rate	10^{-4}
Grad norm clip	0.25
Weight decay	10^{-4}
Learning rate decay	Linear warmup for first 10^5 training steps
Context Length (N/A to BC)	20
Maximum Episode length	1000
Learning Rate	$6e-4$
Number of workers	16
Number of evaluation episodes	100
Number of iterations	10
Steps per iteration	5000

Table 14: Hyperparameters used with DT and BC on D4RL.

Hyperparameter	Value
Reference implementation	https://github.com/tinkoff-ai/CORL/tree/main (Apache 2.0 License)
Batch Size	2048
Steps per Iteration	1250
Number of Iterations	100
Discount	0.99
Alpha multiplier	1
Policy Learning Rate	3e-4
QF Learning Rate	3e-4
Soft target update rate	5e-3
BC steps	100k
Target update period	1
CQL n_actions	10
CQL importance sample	True
CQL lagrange	False
CQL target action gap	-1
CQL temperature	1
CQL min q weight	5

Table 15: Hyperparameters used with CQL on D4RL. We used identical hyperparameters as the reference implementation.

Hyperparameter	**VALUE**
Reference implementation	https://github.com/denisyarats/exorl (MIT License)
BC hidden dim	1024
BC batch size	1024
BC LR	1e-4
CQL LR	1e-4
CQL Critic Target tau	0.01
CQL Critic Lagrange	False
CQL target penalty	5
CQL Batch size	1024

Table 16: Hyperparameters used with CQL on EXORL. We used identical hyperparameters as the reference implementation.

Hyperparameter	**VALUE**
Reference implementation	https://github.com/ARISE-Initiative/robomimic (MIT License)
BC LR	1e-4
BC Learning Rate Decay Factor	0.1
BC encoder layer dimensions	[300, 400]
BC decoder layer dimensions	[300, 400]
CQL discount	0.99
CQL Q-network LR	1e-3
CQL Policy LR	3e-4
CQL Actor MLP dimensions	[300-400]
CQL Lagrange threshold τ	5

Table 17: Hyperparameters used with BC and CQL on ROBOMIMIC. We used identical hyperparameters as the reference implementation.

H ABLATION STUDY TO DETERMINE THE IMPORTANCE OF ARCHITECTURAL COMPONENTS OF DT

In this section, we present the results of our ablation study, which was conducted to assess the significance of various architectural components of the DT. To isolate the impact of individual hyperparameters, we altered one at a time while keeping all others constant. Our findings indicate that the ATARI benchmark is better suited for examining scaling trends compared to the D4RL benchmark. This is likely due to the bounded rewards present in D4RL tasks, which may limit the ability to identify meaningful trends. We did not observe any significant patterns in the D4RL context. A key insight from this investigation is that the performance of DT, when averaged across Atari games, improved as we increased the number of attention heads. However, we did not notice a similar trend when scaling the number of layers (Figure 4). It is also important to mention that the original DT study featured two distinct implementations of the architecture. The DT variant used for reporting results on ATARI benchmark had 8 heads and 6 layers, while the one employed for D4RL featured a single head and 3 layers.

Number of Heads	BREAKOUT	QBERT	SEAQUEST	PONG
4	267.5 ± 97.5	15.4 ± 11.4	2.5 ± 0.4	106 ± 8.1
8	155.01 ± 49.86	32.05 ± 14.23	2.08 ± 0.44	90.2 ± 11.33
16	220.67 ± 51.02	23.74 ± 10.75	1.94 ± 0.39	54.32 ± 44
32	246.2 ± 62.8	34.27 ± 11.8	1.97 ± 0.38	72.91 ± 41.36
64	249.54 ± 54.13	24.53 ± 11.63	2.11 ± 0.56	69.76 ± 34.49

Table 18: Ablation study to determine the importance of the number of heads on the performance of DT on the ATARI benchmark. The experiment was conducted with fully mixed Data (50 buffers) with 10 evaluation rollouts (number of layers=6).

Number of Layers	BREAKOUT	QBERT	SEAQUEST	PONG
6	229.58 ± 49.86	24 ± 9.28	2.08 ± 0.44	90.2 ± 11.33
8	160.8 ± 44.1	32.34 ± 14.07	1.9 ± 0.36	73.35 ± 40.4
12	226.25 ± 79.47	24.74 ± 12.61	3.75 ± 0.78	94.23 ± 22.44
16	218.55 ± 58.92	32.04 ± 8.49	2.28 ± 0.59	55.95 ± 48.55
24	159.02 ± 49.35	40.68 ± 13.99	4.83 ± 1.3	95.75 ± 8.36
32	239.31 ± 71.21	23.22 ± 8.15	2.29 ± 0.46	86 ± 29.72
64	223.71 ± 59.65	31.63 ± 12.56	2.44 ± 0.61	73.6 ± 39.68

Table 19: Ablation study to determine the importance of the number of layers on the performance of DT on the ATARI benchmark. The experiment was conducted with fully mixed Data (50 buffers) with 10 evaluation rollouts (number of heads=8).

Embedding dimension	BREAKOUT	QBERT	SEAQUEST	PONG
128	229.58 ± 49.86	24 ± 9.28	2.08 ± 0.44	90.2 ± 11.33
256	289 ± 76.79	32.34 ± 14.07	1.91 ± 0.34	39.25 ± 48.09
512	216.17 ± 49.13	25.59 ± 9.17	-	2.08 ± 0.59
1024	59.7 ± 78.78	3.93 ± 8.44	0.11 ± 0.06	-

Table 20: Ablation study to determine the importance of embedding dimension on the performance of DT on the ATARI benchmark. The experiment was conducted with fully mixed Data (50 buffers) with 10 evaluation rollouts (number of heads=8).

Context Length	BREAKOUT	QBERT	SEAQUEST	PONG
30	229.58 ± 49.86	24 ± 9.28	2.08 ± 0.44	61.04 ± 38.47
32	227.51 ± 68.3	25.27 ± 8.09	2.16 ± 0.32	-
36	213.19 ± 66.82	23.43 ± 15.31	2.4 ± 0.49	-
42	219.98 ± 53.5	25.58 ± 9.86	1.99 ± 0.35	-
48	263.29 ± 49.68	22.91 ± 15.13	1.96 ± 0.37	-

Table 21: Ablation study to determine the importance of context length on the performance of DT on the ATARI benchmark. The experiment was conducted with fully mixed Data (50 buffers) with 10 evaluation rollouts (number of heads=8, num of layers=6).

Heads	BREAKOUT	QBERT	SEAQUEST	PONG
4	194.15 ± 36.66	20.88 ± 10.05	1.94 ± 0.37	86.96 ± 16.67
8	212.58 ± 38.55	29.24 ± 10.83	2.08 ± 0.31	52.95 ± 42.54
16	236.68 ± 31.73	29.52 ± 8.37	1.94 ± 0.3	85.57 ± 14.29
32	221.17 ± 23.81	23.34 ± 6.34	1.73 ± 0.24	90.75 ± 10.99
64	-	35.55 ± 10.86	2.01 ± 0.34	63.36 ± 36.23

Table 22: Ablation study to determine the importance of heads on DT performance on the ATARI benchmark. The experiment was conducted with fully mixed Data (50 buffers) with 100 evaluation rollouts (number of layers=6).

Layers	BREAKOUT	QBERT	SEAQUEST	PONG
6	212.58 ± 38.55	29.24 ± 10.83	2.08 ± 0.31	70.91 ± 34.63
8	220.35 ± 45	32.42 ± 13.79	2.12 ± 0.59	70.4 ± 36.28
12	206.83 ± 46.2	19.48 ± 7.46	2.06 ± 0.4	90.77 ± 12.91
16	212.67 ± 36.98	29.04 ± 10.82	2.22 ± 0.35	73.84 ± 37.41
24	173.43 ± 56.4	21.41 ± 10.1	2.11 ± 0.48	87.5 ± 15.81
32	206.07 ± 45.69	32.15 ± 9.07	2.18 ± 0.44	66.4 ± 40.82
64	177.64 ± 102.81	27.74 ± 5.97	2.04 ± 0.42	35.98 ± 43.27

Table 23: Ablation study to determine the importance of layers on the performance of DT on the ATARI benchmark. The experiment was conducted with fully mixed Data (50 buffers) with 100 evaluation rollouts (number of heads=8).

Heads	BREAKOUT	QBERT	SEAQUEST	PONG
1	142.6 ± 53.85	30.93 ± 18.4	3.5 ± 0.81	85.74 ± 25.64
4	155.01 ± 43.58	33.26 ± 11.12	4.37 ± 1.25	85.04 ± 16.68
8	188.38 ± 67.43	33.94 ± 12.46	3.53 ± 0.73	65.9 ± 38.08
16	176.58 ± 69.9	25.32 ± 9.71	3.58 ± 0.96	75.35 ± 37.8
32	225.85 ± 68.21	27.9 ± 6.47	3.21 ± 0.73	73.7 ± 37.27
64	246.62 ± 120.95	37.25 ± 20.03	3.23 ± 0.87	72.14 ± 37.78

Table 24: Ablation study to determine the importance of heads on the performance of DT on the ATARI benchmark. The experiment was conducted with the last 10 buffers (out of 50) with 100 evaluation rollouts (number of layers=6).

Layers	BREAKOUT	QBERT	SEAQUEST	PONG
1	151.71 ± 37.17	21.49 ± 8.54	3.65 ± 1.12	79.2 ± 26.77
4	186.49 ± 59.76	28.04 ± 16.35	3.73 ± 1	73.55 ± 37.41
6	188.38 ± 67.43	33.94 ± 12.46	3.53 ± 0.73	65.9 ± 38.08
8	160.8 ± 44.1	22.52 ± 14.71	3.31 ± 1.13	74.11 ± 37.98
12	225.85 ± 68.21	27.9 ± 6.47	3.21 ± 0.73	73.7 ± 37.27
16	246.62 ± 120.95	37.25 ± 20.03	3.23 ± 0.87	72.14 ± 37.78
24	159.02 ± 49.35	40.68 ± 13.99	4.83 ± 1.3	70.4 ± 35.52

Table 25: Ablation study to determine the importance of layers on the performance of DT on the ATARI benchmark. The experiment was conducted with the last 10 buffers (out of 50) with 100 evaluation rollouts (number of heads=8).

Heads	HALF CHEETAH	HOPPER	WALKER2D
4	42.74 (0.2)	77.08 (7)	74.64 (1.6)
8	42.62 (0.2)	72.3 (3.1)	74.08 (0.8)
16	42.79 (0.2)	74.07 (3.4)	73.59 (1.3)
32	42.53 (0.2)	72.76 (1.1)	76.17 (1.4)
64	42.51 (0.1)	74.14 (1.7)	74.51 (2.4)

Table 26: Ablation study to determine the importance of heads on the performance of DT the D4RL medium tasks. The results are averaged across 100 evaluation rollouts (number of layers=6).

Heads	HALF CHEETAH	HOPPER	WALKER2D
4	37.87 (0.3)	89.06 (0.4)	67.93 (2.7)
8	38.07 (0.3)	91.32 (3.6)	72.7 (2.7)
16	38.46 (0.4)	88.88 (1.3)	72.16 (1)
32	37.88 (0.3)	88.15 (2.9)	72.65 (2.8)
64	38.17 (0.5)	90.29 (3.5)	71.99 (5)

Table 27: Ablation study to determine the importance of heads on the performance of DT on the D4RL medium-replay tasks. The results are averaged across 100 evaluation rollouts (number of layers=6).

Heads	HALF CHEETAH	HOPPER	WALKER2D
4	91.52 (0.7)	110.58 (0.1)	108.57 (0.3)
8	90.55 (0.9)	110.18 (0.4)	108.69 (0.3)
16	90.61 (0.7)	110.79 (0.3)	108.28 (0)
32	92.06 (0.2)	110.84 (0.3)	108.16 (0)
64	91.35 (0.3)	110.87 (0.4)	108.45 (0.1)

Table 28: Ablation study to determine the importance of heads on the performance of DT on the D4RL medium-expert tasks. The results are averaged across 100 evaluation rollouts (number of layers=6).

Layers	HALF CHEETAH	HOPPER	WALKER2D
4	42.51 (0)	72.05 (1.8)	74.68 (0.6)
6	42.62 (0.2)	73.37 (4.4)	74.08 (0.8)
8	42.48 (0)	73.25 (5.2)	75.04 (1.9)
12	42.55 (0.2)	70.37 (2.6)	74.49 (0.89)
16	42.45 (0)	63.3 (1.7)	74.53 (1.9)
24	42.4 (0.1)	69.07 (4.35)	75.64 (1.4)

Table 29: Ablation study to determine the importance of layers on the performance of DT on the D4RL medium tasks. The results are averaged across 100 evaluation rollouts (number of heads=8).

Layers	HALF CHEETAH	HOPPER	WALKER2D
4	38.8 (0.7)	92.09 (3)	69.33 (4.2)
6	38.07 (0.3)	91.32 (3.6)	72.2 (2.7)
8	37.14 (1.3)	90.71 (0.64)	70.14 (2.37)
12	37.29 (1)	89.11 (2.1)	69.36 (2.3)
16	37.15 (0.4)	88.6 (3.3)	71.4 (1.8)
24	37.46 (0.7)	83.17 (4.6)	73.66 (2.4)
32	38.39 (0.9)	87.94 (1.6)	69.43 (0.9)
64	38.09 (0.7)	81.95 (0.46)	68.01 (2.28)

Table 30: Ablation study to determine the importance of layers on the performance of DT on the D4RL medium-replay tasks. The results are averaged across 100 evaluation rollouts (number of heads=8).

Layers	HALF CHEETAH	HOPPER	WALKER2D
4	92.21 (0.2)	110.26 (0.6)	108.33 (0.6)
8	90.55 (0.9)	110.18 (0.4)	108.37 (0)
16	90.52 (0.8)	110.49 (0.2)	108.79 (0.1)
32	90.06 (0.6)	110.8 (0.2)	108.61 (0.4)
64	90.59 (0.6)	110.46 (0.4)	108.82 (0)

Table 31: Ablation study to determine the importance of layers on the performance of DT on the D4RL medium-expert tasks. The results are averaged across 100 evaluation rollouts (number of heads=8).

Context Length	HALF CHEETAH	HOPPER	WALKER2D
10	42.68 (0)	67.6 (3.3)	75.4 (1.31)
20	42.71 (0.1)	71 (5.4)	75.25 (0.9)
30	42.72 (0.1)	81.55 (10.2)	75.8 (0.2)
40	42.67 (0.2)	73.88 (0.9)	74.73 (1.8)
50	42.66 (0)	83.14 (5.3)	72.52 (0.7)
60	42.49 (0)	85.54 (6.9)	74.18 (0.3)
70	42.54 (0.1)	83.75 (0.03)	74.04 (2.7)

Table 32: Ablation study to determine the importance of context length on the performance of DT on the D4RL medium tasks. The results are averaged across 100 evaluation rollouts (number of layers=6, number of heads=8).

Context Length	HALF CHEETAH	HOPPER	WALKER2D
10	36.98 (0.2)	80.07 (1.1)	69.73 (2.2)
20	37.7 (0.1)	84.28 (2.6)	67.62 (2.6)
30	35.71 (0.1)	91.37 (3.9)	68.71 (1.4)
40	35.34 (0.6)	87.08 (1.5)	69.33 (2.4)
50	35.36 (0.5)	87.99 (2.3)	64.61 (5.2)
60	37.19 (0.5)	86.82 (3.5)	64.55 (1.2)
70	36.62 (1.4)	90.27 (0.8)	63.48 (2.5)

Table 33: Ablation study to determine the importance of context length on the performance of DT on the D4RL medium-replay tasks. The results are averaged across 100 evaluation rollouts (number of layers=6, number of heads=8).

Context Length	HALF CHEETAH	HOPPER	WALKER2D
10	87.61 (2.64)	109.99 (0.48)	108.97 (0.92)
20	86.21 (3.75)	109.54 (0.72)	107.97 (0.82)
30	88.24 (2.17)	110.16 (0.48)	108.24 (0.26)
40	87.38 (1.15)	110.71 (0.4)	108.6 (0.04)
50	88.22 (0.96)	110.74 (0.41)	107.89 (0.56)

Table 34: Ablation study to determine the importance of context length on the performance of DT on the D4RL medium-expert tasks. The results are averaged across 100 evaluation rollouts (number of layers=6, number of heads=8).

Context Length	HALF CHEETAH	HOPPER	WALKER2D
32	86.27 (0.3)	108.92 (0.5)	108.18 (0.3)
64	86.62 (2)	110.12 (0.7)	107.85 (0.5)
128	89.15 (1.2)	109.47 (0.3)	108.53 (0)
256	90.98 (0.9)	109.47 (1.1)	107.73 (0)
512	91.3 (0.3)	109.61 (0.7)	107.72 (0)
1024	91.38 (0.4)	109.04 (0.8)	108.42 (0.2)

Table 35: Ablation study to determine the importance of context length on the performance of DT on the D4RL medium-replay tasks. The results are averaged across 100 evaluation rollouts (number of layers=6, number of heads=8).

I DT ON EXORL

We additionally conduct smaller-scale experiments in EXORL, which allows us to study the performance of DT on reward-free play data. Typical offline RL datasets are collected from a behavior policy that aims to optimize some (unknown) reward. Contrary to this practice, the **EXORL** benchmark (Yarats et al., 2022) was obtained from reward-free exploration. After the acquisition of an (s, a, s') dataset, a reward function is chosen and used to include rewards in the data. This same reward function is used during evaluation. We consider the WALKER WALK, WALKER RUN, and WALKER STAND environments (APT). All scores are averaged over 10 evaluation episodes.

In the following section, we present the results obtained using DT on three distinct environments from the EXORL framework. `Returns-to-go` in the tables presented below represent returns-to-value provided to DT at the time of inference. Upon comparing the metrics in the EXORL study, we noticed that DT's performance falls short compared to CQL, which may be attributed to the data being collected in a reward-free setting. Although investigating the behavior of these agents in reward-free settings presents an avenue for future research, we propose the following hypothesis. Typically, the exploration of new states in a reward-free environment is conducted through heuristics such as curiosity (ICM) (Pathak et al., 2017) or entropy maximization (APT) (Liu and Abbeel, 2021). The reward functions defined by these heuristics differ from those used for relabeling data when training offline RL agents. Consequently, bootstrapping-based methods might be better equipped to learn a mapping between the reward function determined by the heuristic and the one employed for data relabeling.

LR	Returns-to-Go						
	200	**300**	**400**	**500**	**600**	**700**	**800**
6e-4	123.28 (1.9)	120.18 (9.3)	126.46 (2.1)	131.82 (7.4)	128.47 (4.1)	130.95 (9.1)	125.61 (7.4)
1e-3	119.25 (1.2)	121.59 (5.1)	128.85 (5)	132.35 (6.5)	136.59 (10.7)	124.58 (7.7)	121.68 (5.8)
3e-3	123.68 (5.3)	123.43 (5.3)	128.23 (0.9)	122.54 (8.9)	127.24 (2.2)	116.33 (1.9)	122.34 (6.3)
5e-3	110.35 (7.3)	120.65 (8.1)	123.82 (11.4)	117.38 (6.7)	120.45 (2.8)	112.27 (11.5)	104.63 (8.1)
7e-3	120.6 (7.8)	125.98 (2.8)	117.76 (11)	123.76 (14.9)	121.7 (6.4)	110.95 (6.7)	120.18 (6.9)
9e-3	125.04 (2.5)	114.34 (5.62)	116.57 (5.8)	115.17 (1.6)	123.71 (15.8)	124.65 (6.8)	123.9 (5.2)
1-e2	126.93 (8.2)	125.18 (3.9)	130 (4.2)	118.92 (10.8)	107.87 (10.4)	126.28 (8.7)	119.57 (13)

Table 36: Performance of DT on WALKER WALK from EXORL as returns-to-go value is varied. The experiment was run with the following parameters: batch size=7200, warmup steps=10, 200k timestep data, 150 gradient updates, and context length: 20.

LR	Returns-to-Go						
	200	**300**	**400**	**500**	**600**	**700**	**800**
6e-4	133.68 (10.9)	139.51 (5.1)	137.5 (12.3)	131.2 (3.8)	130.14 (6.5)	129.8 (1.4)	129.79 (3.9)
1e-3	131.62 (8.8)	126.34 (5.6)	125.71 (4.4)	129.58 (9.2)	132.2 (11.4)	127.94 (8.4)	126.6 (9.5)
3e-3	126.71 (3.3)	129.85 (1.6)	119.36 (3.7)	125.56 (1.6)	128 (12.4)	125.86 (4.4)	118.13 (5)
5e-3	123.05 (4)	119.7 (6.5)	124.79 (7.4)	124.48 (8.3)	123.24 (12.3)	122.25 (8.1)	129.83 (4.2)
7e-3	108.4 (2.5)	116.85 (6.9)	120.8 (10.5)	112.59 (9.1)	116.12 (3.2)	113.55 (3.5)	117.96 (1.8)
9e-3	107.79 (7.4)	98.57 (18)	110.52 (31.5)	102.2 (9.6)	101.21 (20.4)	102.91 (20.1)	90.19 (21.9)
1-e2	112.69 (10.5)	100.42 (13.7)	114 (24.3)	118.13 (3.6)	105.55 (9.9)	108.74 (14.9)	126.21 (16.2)

Table 37: Performance of DT on WALKER WALK from EXORL. The experiment was run with the following parameters: batch size=7200, warmup steps=30, 200k timestep data, 150 gradient updates, and context length: 20.

LR	Warmup	Returns-to-Go						
		200	**300**	**400**	**500**	**600**	**700**	**800**
6e-4	30	133.68 (10.9)	139.51 (5.1)	137.5 (12.3)	131.2 (3.8)	130.14 (6.5)	129.8 (1.4)	129.79 (3.9)
4e-4	30	136.16 (4.7)	131.23 (6.5)	131.67 (5.2)	121.06 (6.2)	121.95 (3.5)	129.54 (0.1)	137.26 (0.9)
1e-4	30	117.7 (4.3)	124.08 (9.7)	114.02 (1.9)	118.9 (3.8)	119.09 (0.7)	123.83 (9.7)	120.86 (7.0)
6e-4	50	130.14 (11.2)	129.53 (8.2)	136.92 (7.4)	123.96 (0.4)	132.93 (1)	123.17 (5.5)	136.91 (9.6)
6e-4	80	125.44 (1)	131.98 (2.3)	129.48 (7.8)	123.6 (6.8)	127.62 (2.9)	127.64 (6.7)	123.53 (3.1)
6e-4	100	124.18 (4.5)	128.26 (2)	122.23 (6.8)	127.9 (0.8)	125.78 (3.7)	131.39 (5.9)	131.83 (5.1)

Table 38: Performance of DT on WALKER WALK from EXORL. The experiment was run with the following parameters: batch size=7200, 200k timestep data, 150 gradient updates, and a context length of 20.

LR	Warmup	Returns-to-Go						
		200	**300**	**400**	**500**	**600**	**700**	**800**
6e-4	50	123.53 (13)	127.81 (5.3)	121.44 (6.7)	112.79 (5.6)	123.33 (2.7)	120.44 (1.9)	124.58 (7)
6e-4	100	117.02 (4.3)	137.5 (6.5)	126.24 (8.1)	124.64 (6.8)	125.43 (9.9)	115.1 (2.0)	125.67 (7.8)
7e-4	100	120.91 (9.1)	128.69 (3.4)	121.03 (4.4)	112.11 (6)	120.67 (4.5)	127.57 (3.8)	120.05 (5.9)
1e-3	30	123.64 (1.3)	118.84 (11.7)	129.66 (7.5)	126.77 (5.5)	122.59 (7.6)	127.9 (5.6)	127.19 (12.3)

Table 39: Performance of DT on WALKER WALK from EXORL. The experiment was run with the following parameters: batch size=7200, 200k timestep data, 300 gradient updates, and a context length of 20.

LR	Warmup	Returns-to-Go						
		200	**300**	**400**	**500**	**600**	**700**	**800**
6e-4	50	57.56 (0.8)	57.49 (1.4)	56.25 (1.3)	56.38 (1.1)	57.4 (1)	57.87 (1.6)	56.23 (2)
6e-4	100	56.99 (0.6)	58.64 (0.9)	57.88 (1.3)	57.15 (1.3)	60.11 (2.6)	57.25 (2.1)	57.7 (0.8)
8e-4	40	59.34 (1.8)	57.98 (2.4)	57.82 (1.4)	56.9 (2.2)	56.48 (2.4)	60.36 (2.3)	58.5 (2.1)
6e-4	200	56.47 (2.5)	59.69 (2.8)	56.6 (0.7)	59.36 (1.0)	57.33 (2.0)	59.18 (0.1)	58.36 (0.1)
7e-4	100	60.3 (0.1)	57.12 (0.7)	57.03 (2.0)	56.32 (0.7)	59.55 (1.2)	58.2 (1.3)	58.6 (2.2)
1e-3	30	62.48 (3.3)	59.23 (1.8)	56.62 (1.1)	58.69 (1.7)	58.94 (2.6)	56.25 (1.5)	57.55 (1.7)

Table 40: Performance of DT on WALKER RUN from EXORL. The experiment was run with the following parameters: batch size=7200, 200k timestep data, 300 gradient updates, context length: 20.

LR	Warmup	Returns-to-Go						
		200	**300**	**400**	**500**	**600**	**700**	**800**
6e-4	100	263.71 (6.8)	292.08 (6.3)	278.26 (14.7)	271.19 (3.8)	283.36 (11.1)	284.05 (15.2)	279.87 (8.8)
8e-4	40	270.83 (1.7)	271.04 (14.6)	275.7 (5.5)	273.92 (13.9)	269.5 (0.7)	290.78 (14.7)	286.77 (6.8)
6e-4	200	266.37 (14.8)	271.16 (7.2)	278.27 (8.2)	291.59 (9.4)	287.37 (4.4)	274.2 (6.9)	272.6 (5.7)

Table 41: Performance of DT on WALKER STAND from EXORL. The experiment was run with the following parameters: batch size=7200, 200k timestep data, 300 gradient updates, context length: 20.

LR	Returns-to-Go						
	200	**300**	**400**	**500**	**600**	**700**	**800**
1e-4	128.52 (3)	125.83 (3.2)	116.24 (2.5)	130.23 (12.7)	137.82 (3.7)	126.32 (3.4)	125 (5.5)
6e-4	114.56 (11.9)	107.41 (5.5)	127.49 (10.4)	115.08 (8.4)	99.1 (4.0)	117.1 (7.3)	97.73 (1.3)
9e-4	106.24 (7.5)	102.84 (16.0)	102.74 (14.7)	95.55 (5.5)	110.13 (10.4)	105.28 (13.3)	100.03 (7.0)
1e-3	92.71 (3.1)	104.86 (15.3)	99.35 (1.5)	96.8 (8.0)	86.95 (10.0)	98.01 (14.7)	92.7 (1.6)
4e-3	45.65	42.86	39.6	41.93	39.71	39.88	43.05
8e-3	56.3	39.3	44.63	48.93	39.35	44.12	52.88

Table 42: Performance of DT on WALKER WALK from EXORL. The experiment was run with the following parameters: batch size=7200, warmup steps=100, 200k timestep data, 900 gradient updates.

LR	Returns-to-Go						
	200	**300**	**400**	**500**	**600**	**700**	**800**
1e-4	113.53 (2.5)	123.63 (2.7)	115.69 (2.0)	116.66 (1.7)	114.54 (1.1)	115.48 (2.1)	122.86 (6.4)
6e-4	103.02 (5.5)	102.17 (1.3)	94.05 (0.6)	98.45 (0.0)	95.22 (1.5)	101.86 (8.9)	114.71 (3.8)
9e-4	92.71 (8.7)	92.18 (0.8)	102.49 (5.0)	88.61 (11.8)	92.95 (3.2)	94.97 (2.2)	97.31 (3.6)
1e-3	84.65	93	97.18	93.57	95.27	92.04	88.7
4e-3	57.96 (1.0)	66.73 (2.9)	61.93 (14.4)	46.71 (3.4)	58.77 (9.0)	51.75 (0.2)	47.44 (5.1)
8e-3	39.59 (2.4)	38.9 (4.6)	44.71 (12.2)	41.27 (2.9)	40.34 (5.8)	48.63 (4.7)	45.2 (7.2)

Table 43: Performance of DT on WALKER WALK from EXORL. The experiment was run with the following parameters: batch size=7200, 200k timestep data, warmup steps=100, 900 gradient updates, context length=10.

LR	Returns-to-Go						
	200	**300**	**400**	**500**	**600**	**700**	**800**
1e-4	113.53 (2.5)	123.63 (2.7)	115.69 (2.0)	116.66 (1.7)	114.54 (1.1)	115.48 (2.1)	122.86 (6.4)
6e-4	103.02 (5.5)	102.17 (1.3)	94.05 (0.6)	98.45 (0.0)	95.22 (1.5)	101.86 (8.9)	114.71 (3.8)
9e-4	92.71 (8.7)	92.18 (0.8)	102.49 (5.0)	88.61 (11.8)	92.95 (3.2)	94.97 (2.2)	97.31 (3.6)
1e-3	84.65	93	97.18	93.57	95.27	92.04	88.7
4e-3	57.96 (1.0)	66.73 (2.9)	61.93 (14.4)	46.71 (3.4)	58.77 (9.0)	51.75 (0.2)	47.44 (5.1)
8e-3	39.59 (2.4)	38.9 (4.6)	44.71 (12.2)	41.27 (2.9)	40.34 (5.8)	48.63 (4.7)	45.2 (7.2)

Table 44: Performance of DT on WALKER WALK from EXORL. The experiment was run with the following parameters: batch size=7200, 200k timestep data, warmup steps=100, 900 gradient updates, context length=10.

LR	Returns-to-Go						
	200	**300**	**400**	**500**	**600**	**700**	**800**
1e-4	116.61 (7.8)	111.97 (6.3)	114.26 (4.4)	116.88 (4.9)	123.97 (3.4)	111.21 (3.7)	108.62 (2.5)
6e-4	112.06 (6.3)	113.17 (2.5)	104.78 (3.3)	112.57 (6.7)	107.5 (3.4)	112.69 (4.6)	106.85 (2.8)
9e-4	108.42 (5.7)	102.44 (2.3)	99.46 (8.1)	105.42 (1.7)	106.62 (9.3)	99.2 (6.2)	102.55 (8.4)
1e-3	103.74 (6.5)	106.42 (1.8)	98.7 (6.2)	103.34 (4.2)	116.92 (3.4)	107.22 (7.0)	105.08 (11.6)
4e-3	87.2 (6.4)	83.95 (4.2)	89.53 (8.1)	77.35 (12.4)	90.34 (2.6)	89.65 (8)	93.22 (7.8)
8e-3	57.73 (10.6)	59.11 (9.3)	56.23 (4.2)	58.82 (12.9)	52.2 (3.6)	62.79 (2.4)	54.29 (9.0)

Table 45: Performance of DT on WALKER WALK (Proto) from EXORL. The experiment was run with the following parameters: batch size=7200, 200k timestep data, 900 gradient updates, context length=10.

LR	Returns-to-Go						
	200	**300**	**400**	**500**	**600**	**700**	**800**
1e-4	116.61 (7.8)	111.97 (6.3)	114.26 (4.4)	116.88 (4.9)	123.97 (3.4)	111.21 (3.7)	108.62 (2.5)
6e-4	112.06 (6.36)	113.17 (2.5)	104.78 (3.3)	112.57 (6.7)	107.5 (3.4)	112.69 (4.6)	106.85 (2.8)
9e-4	108.42 (5.7)	102.44 (2.3)	99.46 (8.1)	105.42 (1.7)	106.62 (9.3)	99.2 (6.2)	102.55 (8.4)
1e-3	103.74 (6.5)	106.42 (1.8)	98.7 (6.2)	103.34 (4.2)	116.92 (3.4)	107.22 (7.0)	105.08 (11.6)
4e-3	87.2 (6.4)	83.95 (4.2)	89.53 (8.1)	77.35 (12.4)	90.34 (2.6)	89.65 (8.0)	93.22 (7.8)
8e-3	57.73 (10.6)	59.11 (9.3)	56.23 (4.2)	58.82 (12.9)	52.2 (3.6)	62.79 (2.4)	54.29 (9.0)

Table 46: Performance of DT on WALKER WALK (Proto) from EXORL. The experiment was run with the following parameters: batch size=7200, 200k timestep data, 900 gradient updates, context length=10.