Should We Ever Prefer Decision Transformer for Offline Reinforcement Learning?

Anonymous authors
Paper under double-blind review

Abstract

In recent years, extensive work has explored the application of the Transformer architecture to reinforcement learning problems. Among these, Decision Transformer (DT) (Chen et al., 2021) has gained particular attention in the context of offline reinforcement learning due to its ability to frame return-conditioned policy learning as a sequence modeling task. Most recently, Bhargava et al. (2024) provided a systematic comparison of DT with more conventional MLP-based offline RL algorithms, including Behavior Cloning (BC) and Conservative Q-Learning (CQL), and claimed DT exhibits superior performance in sparse-reward and low-quality data settings.

In this paper, through experimentation on robotic manipulation tasks (Robomimic) and locomotion benchmarks (D4RL), we show that MLP-based Filtered Behavior Cloning (FBC) achieves competitive or superior performance compared to DT in sparse-reward environments. FBC simply filters out low-performing trajectories from the dataset and then performs ordinary behavior cloning on the filtered dataset. FBC is not only very straightforward, but it also requires less training data and is computationally more efficient. The results therefore suggest that DT is not preferable for sparse-reward environments. From prior work, arguably, DT is also not preferable for dense-reward environments. Thus, we pose the question: Is DT ever preferable?

1 Introduction

The transformer architecture (Vaswani et al., 2017), originally introduced for natural language processing, has rapidly become a foundational model across multiple domains, including a broad range of NLP and computer vision tasks. In recent years, there has been growing interest in the field of reinforcement learning (RL) in integrating transformers into the policy learning pipeline (Li et al., 2023), notably Decision Transformer (Chen et al., 2021), and Trajectory Transformer (Janner et al., 2021). Among these, Decision Transformer (DT) has garnered particular attention in the context of offline reinforcement learning due to its ability to frame return-conditioned policy learning as a sequence modeling task.

Recently, Bhargava et al. (2024) provided a thorough comparison of DT with conventional MLP-based offline RL algorithms, including Behavior Cloning (BC) and Conservative Q-Learning (CQL) (Kumar et al., 2020). They primarily examined two offline datasets: the D4RL (Fu et al., 2020) dataset and the Robomimic (Zhu et al., 2020b) dataset. For both datasets, they consider both dense reward and sparse reward variations. In the case of sparse rewards, through extensive experimentation, they argue that DT is preferable to conventional offline algorithms such as BC and CQL. For the dense-reward versions, their experiments show that CQL is preferable to DT for the D4RL dataset, and that DT is preferable to CQL for the Robomimic datasets.

In this paper, we reexamine the question of whether DT is preferable for sparse-reward environments. To this end, we present a simple yet effective MLP-based algorithm, Filtered Behavior

Cloning (FBC). FBC simply filters out low-performing trajectories from the dataset and then performs ordinary behavior cloning with the remaining trajectories. On robotic manipulation tasks (Robomimic) and locomotion benchmarks (D4RL), we demonstrate that FBC with a small multi-layer perceptron (MLP) backbone achieves competitive or superior performance compared to DT in sparse-reward environments. For example, for the D4RL datasets with sparsification, FBC beats DT for 7 of the 9 datasets, and improves aggregate performance by about 4%. For the Robomimic dataset, FBC beats DT for both of the two datasets considered in Bhargava et al. (2024), and provides an aggregate performance improvement of about 3.5%. Moreover, FBC uses fewer parameters, uses less training data, and learns more quickly in terms of wall-clock time. To make a fair comparison, all of the experiments reported in this paper follow closely the setups in Bhargava et al. (2024), including the choice of datasets, the DT architectures and their sizes, and the hyperparameters.

Additionally, we also consider Filtered Decision Transformer (FDT), where DT learns from the same filtered dataset as FBC does. Here we find that for the D4RL benchmarks, FBC performs better than FDT, and for the Robomimic benchmark, FBC and FDT perform about the same.

Although we do not study dense-reward environments in this paper, based on prior literature (Emmons et al., 2021; Tarasov et al., 2023; Yamagata et al., 2023; Hu et al., 2024), DT is arguably also not preferable to conventional MLP-based offline algorithms for dense-reward datasets in general. Thus, we pose the question: *Is DT ever preferable for raw-state robotic tasks?*

2 Related Work

In addition to leveraging the attention mechanism of the Transformer to encode sequential observation histories as meaningful representations for downstream decision-making tasks (Tang & Ha, 2021; Guhur et al., 2023; Parisotto et al., 2020; Li et al., 2022), the Transformer architecture itself is capable of formulating and solving decision-making problems as sequence modeling tasks. Some of the major examples include Decision Transformer (DT) (Chen et al., 2021), Trajectory Transformer (TT) (Janner et al., 2021), Generalized Decision Transformer (GDT) (Furuta et al., 2022), Bootstrapped Transformer (BooT) (Wang et al., 2022), Behavior Transformer (BeT) (Shafiullah et al., 2022), and Q-Transformer (Chebotar et al., 2023). These extensions aim to better align transformer models with the structure of decision-making tasks.

Decision Transformer, in particular, has received considerable attention to date with over 2000 citations as of June 2025. DT takes in as input sequences of states, actions, and return-to-goes (RTG), and predicts the next action. Despite the appealing architectural innovation, subsequent studies have exposed two major issues with DT: (1) failure in the face of high environmental stochasticity (Emmons et al., 2021; Paster et al., 2022) and (2) the inability to stitch suboptimal trajectories (Yamagata et al., 2023; Hu et al., 2024).

Recently, Bhargava et al. (2024) provided a thorough comparison of DT with conventional MLP-based offline RL algorithms, including Behavior Cloning (BC) and Conservative Q-Learning (CQL) (Kumar et al., 2020). They primarily examined two offline datasets: the D4RL (Fu et al., 2020) dataset and the Robomimic (Zhu et al., 2020b) dataset. For both datasets, they consider both dense reward and sparse reward variations. They conclude from their experiments that DT is a preferable algorithm in sparse-reward, low-quality data, and long-horizon settings, when compared with vanilla Behavior Cloning (BC) and Conservative Q-learning (CQL) (Kumar et al., 2020).

3 Problem Setup

In this paper, we aim to explore whether DT is preferable in sparse reward environments. Following the experimental design of Bhargava et al. (2024), we consider two sparse-reward settings for our experiments.

Sparse Reward Setting This setting corresponds to environments where binary rewards are assigned only at the terminal timestep of trajectories. Let $\mathcal{D} = \{\tau_i\}_{i=1}^N$ denote a dataset of N trajectories, where each trajectory $\tau_i = \{(s_t^i, a_t^i, r_t^i)\}_{t=1}^{T_i}$ consists of states, actions, and rewards. The reward function is defined as:

$$r_t^i = \begin{cases} 1, & \text{if } t = T_i \text{ and } \tau_i \text{ is successful,} \\ 0, & \text{otherwise.} \end{cases}$$

For example, in a dataset with $N = 300$ trajectories, if 100 are labeled successful, then each of these 100 trajectories receives a cumulative reward of 1; the remaining 200 receive 0.

Sparsified Reward Setting This setting is derived from environments that originally provide dense per-timestep rewards. In offline reinforcement learning, where agents train on static datasets without further interaction with the task environment, sparse-reward conditions can be simulated by post-processing the dataset. Specifically, following Bhargava et al. (2024), we set all intermediate rewards to zero and move the total return of each trajectory to the final timestep. That is, for a trajectory with rewards $r_0, r_1, \ldots, r_{T-1}$, we modify it such that $r_t \leftarrow 0$ for all $t < T-1$, and $r_{T-1} \leftarrow \sum_{t=0}^{T-1} r_t$. This allows the offline learning agent to experience a reward distribution analogous to that of truly sparse environments. The key distinction is that in sparse settings, terminal rewards are inherently binary (typically 0 or 1), whereas in sparsified settings, terminal rewards retain their original non-binary, task-specific values.

4 Methods

Bhargava et al. (2024) compares Behavior Cloning (BC), Conservative Q-Learning (CQL) (Kumar et al., 2020), and Decision Transformer (DT) (Chen et al., 2021). BC and CQL run over an MLP backbone, whereas DT runs over a transformer backbone. We also introduce two new methods: Filtered Behavior Cloning (FBC), which runs over MLP backbones, and Filtered Decision Transformer (FDT), which runs over transformer backbones.

Decision Transformer (DT) During inference, DT chooses the next action based on a fixed-length sequence of target return-to-go, state, and action triplets from prior timesteps. Given a context length K, the action prediction depends on the temporally ordered sequence $\{(R_j, s_j, a_j)\}_{j=t-K+1}^{t}$, where R_j is the target return-to-go. The model processes this sequence through a transformer encoder that uses sinusoidal positional encodings to preserve temporal order, and stacked self-attention layers to model dependencies. One of the critical issues in DT is setting the target return to go. In the sparse-reward setting, setting the target to one is a natural choice. However, in the sparsified-reward setting, it is less obvious.

Filtered Behavior Cloning (FBC) We now consider a very simple algorithm, Filtered Behavior Cloning (FBC). In FBC, we first process the offline dataset by retaining only the high-performing trajectories. *Then we simply apply vanilla BC to the resulting dataset.* When we refer to FBC, we assume that the underlying backbone is a Multi-Layer Perceptron (MLP).

The filtering rule is as follows. For the sparse-reward setting, we only retain the trajectories that are successful. For the sparsified-reward setting, we only retain trajectories for which the final return r_{T-1} is in the top $x\%$ of all the final returns in the dataset. In this paper, we use $x = 10\%$.

Filtered Decision Transformer (FDT). Additionally, we also consider training DT only on the high-performing trajectories, where we filter the trajectories exactly the same way we filter the trajectories for FBC. Although the common belief is that DT learns from both successful and unsuccessful trajectories, including this variant allows us to test whether DT truly benefits from this design in sparse-reward scenarios.

5 Experiments

We evaluate all algorithms introduced in Section 4 for two classes of sparse datasets : (1) a sparsified version of the D4RL dataset (Fu et al., 2020), and (2) the sparse versions of the Robomimic tasks in the Robosuite environment (Mandlekar et al., 2021; Zhu et al., 2020a). Our experimental design closely follows Bhargava et al. (2024) for comparability. Results reported for DT, CQL, and BC were obtained with exactly the same parameters reported in Bhargava et al. (2024). All results are averaged over 5 seeds, and evaluation is conducted every 50 epochs using 50 rollouts. Bolded values indicate the highest-performing method for each task.

5.1 Sparsified D4RL Locomotion

For the D4RL sparsified setting, we evaluate on the locomotion tasks: walker2d, hopper, and halfcheetah, using sparsified versions of the medium, medium-replay, and medium-expert datasets. Further details are included in Appendix A.2.

Table 1: Comparison of BC, CQL, DT, FBC, and FDT across different D4RL environments. Results show the average normalized D4RL score and its standard deviation.

Dataset	**BC**	**CQL**	**DT**	**FBC**	**FDT**
Half Cheetah - Medium	42.76 ± 0.17	38.63 ± 0.81	42.65 ± 1.05	$\mathbf{43.51 \pm 1.35}$	42.17 ± 1.96
Hopper - Medium	64.35 ± 5.6	$\mathbf{73.89 \pm 10.12}$	73.40 ± 11.92	61.12 ± 10.13	63.23 ± 12.28
Walker - Medium	54.62 ± 12.04	19.31 ± 3.17	73.28 ± 11.65	$\mathbf{77.66 \pm 9.61}$	75.24 ± 13.60
Medium Average	53.91 ± 5.93	43.94 ± 4.70	$\mathbf{63.11 \pm 8.21}$	60.76 ± 7.03	60.21 ± 9.28
Half Cheetah - Medium Replay	9.81 ± 9.2	35.00 ± 2.56	39.45 ± 2.43	$\mathbf{41.83 \pm 2.44}$	33.70 ± 7.79
Hopper - Medium Replay	16.19 ± 10.8	83.10 ± 19.21	71.59 ± 10.18	$\mathbf{93.53 \pm 8.05}$	82.74 ± 13.33
Walker - Medium Replay	17.82 ± 4.96	29.02 ± 19.63	65.20 ± 14.31	$\mathbf{72.69 \pm 19.54}$	65.10 ± 17.58
Medium-Replay Average	14.6 ± 8.32	49.04 ± 13.79	58.75 ± 8.97	$\mathbf{69.35 \pm 10.01}$	60.51 ± 12.90
Half Cheetah - Medium Expert	42.95 ± 0.14	24.35 ± 2.38	$\mathbf{93.66 \pm 1.01}$	92.99 ± 0.98	78.24 ± 20.39
Hopper - Medium Expert	62.21 ± 6.5	42.44 ± 12.52	111.37 ± 1.02	$\mathbf{111.46 \pm 0.83}$	106.52 ± 13.28
Walker - Medium Expert	38 ± 9.86	21.30 ± 0.55	107.90 ± 0.98	$\mathbf{109.21 \pm 0.29}$	$\mathbf{109.21 \pm 0.51}$
Medium-Expert Average	47.72 ± 5.5	29.36 ± 5.14	104.31 ± 1.00	$\mathbf{104.55 \pm 0.70}$	97.99 ± 11.39
Total Average	38.74 ± 6.58	40.78 ± 7.88	75.39 ± 6.06	$\mathbf{78.22 \pm 5.91}$	72.90 ± 11.19

Table 1 shows the final performance of the various algorithms. As also reported in Bhargava et al. (2024), we see DT does significantly better than both BC and CQL, confirming that DT is indeed preferable to CQL for the sparsified-reward version of D4RL. *However, we also see that FBC has a higher total average than DT, and that FBC beats DT in 7 of the 9 datasets.* Interestingly, FDT does a little worse than DT (and hence worse than FBC), which seems to indicate that DT can learn from both high-quality and low-quality trajectories.

5.2 Robomimic Sparse Setting

In the Robomimic sparse setting, as in Bhargava et al. (2024), we use the machine-generated (MG) datasets from the Robomimic benchmark built on Robosuite. These datasets are characterized by low-quality demonstrations. Following Bhargava et al. (2024), we consider two tasks: Lift and PickPlaceCan. Details on the tasks and dataset configuration are provided in Appendix A.1.

Table 2: Comparison of BC, CQL, DT, FBC, and FDT methods on the Lift and Can tasks. The best success rate is reported.

Dataset	BC	CQL	DT	FBC	FDT
Lift	0.46 ± 0.05	0.60 ± 0.13	0.92 ± 0.03	$\mathbf{0.94 \pm 0.03}$	0.93 ± 0.05
Can	0.45 ± 0.09	0 ± 0	0.79 ± 0.06	0.84 ± 0.04	$\mathbf{0.89 \pm 0.03}$
Average	0.45 ± 0.07	0.30 ± 0.07	0.86 ± 0.05	0.89 ± 0.04	$\mathbf{0.91 \pm 0.04}$

Table 2 summarizes the best evaluation achieved during training. As also reported in Bhargava et al. (2024), we see DT does significantly better than both BC and CQL, confirming that DT is indeed preferable to CQL for these sparse-reward datasets. *However, we also see that DT, FBC, and FDT all have similar performance, with FBC being a little higher than DT on both datasets, and with FBC also higher than FDT on Lift and lower on Can.*

6 Discussion

We have just established that for the sparse datasets considered in Bhargava et al. (2024), FBC has better performance than DT. We also observed that the training wall-clock time for DT is about three times longer than it is for FBC. Also, the transformer employed for DT has approximately one million parameters, whereas the MLP used for FBC has about half that number. *For sparse reward environments, we therefore conclude that DT is* **not** *preferable.*

Why is it that DT does not beat simple FBC? As a thought experiment, suppose that DT did not have the returns-to-go in training or inference. Also, suppose that the state is Markovian, or can be made nearly Markovian by defining the state as the most recent k states and actions. In this case, we would expect the DT policy to resemble the BC policy obtained with an MLP. In the sparse-reward setting, by including the return-to-go in DT, we are doing no more than indicating to DT which training trajectories are good and which are bad. Thus, intuitively, DT in the sparse reward setting would generate policies that are similar to FBC. In our experiments, we show that DT actually performs somewhat worse than FBC. This could be due to a number of factors, including overfitting and poor credit assignment.

If DT is not preferable for sparse-reward environments, is it preferable for dense-reward? (Bhargava et al., 2024) shows that CQL beats DT for the original (i.e., not sparsified) D4RL datasets. And although we do not provide empirical evidence here, we argue that prior literature implies that DT is also not preferable for diverse dense-reward datasets in general (Emmons et al., 2021; Tarasov et al., 2023; Yamagata et al., 2023; Hu et al., 2024). Thus, we pose the question: *Is DT ever preferable for raw-state robotic tasks?*

7 Conclusion

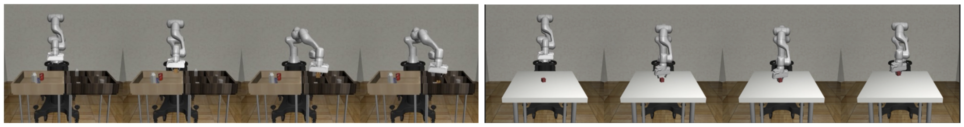
The potential contributions of Decision Transformer (DT) can be broadly categorized into two aspects: first, conditioning on return-to-go (RTG); and second, modeling long-range temporal dependencies via attention mechanisms (Chen et al., 2021). However, increased algorithmic and architectural sophistication does not inherently lead to improved performance in offline reinforcement learning. In sparse-reward domains, where learning signals are limited by nature, the inductive biases built into generic transformer architectures like DT often fail to yield concrete advantages over the basic MLP trained on selectively filtered trajectories.

Our critique is not intended as a wholesale dismissal of transformer-based models. When carefully tailored to the structure of sequential decision-making, transformers can offer genuine benefits. For example, the Behavior Transformer (Shafiullah et al., 2022) incorporates architectural enhancements to better model multimodal behaviors. Similarly, the Graph Decision Transformer (GDT) (Hu et al., 2023) recasts trajectories as causal graphs, mitigating reliance on potentially uninformative RTG signals and achieving strong results on vision-based tasks such as Atari and Gym. More recently, the Q-value Regularized Decision Transformer (QT) (Hu et al., 2024) fuses dynamic programming insights with sequence modeling, consistently outperforming both standard DTs and classical DP-based methods on D4RL benchmarks.

Our empirical analysis, together with Bhargava et al. (2024), calls for ongoing and continuing algorithmic and architectural improvements on top of the vanilla DT to fully unleash the power of the transformer architecture for both dense-reward and sparse-reward RL in future studies.

A Experiment Details

A.1 Robomimic and Robosuite



(a) PickPlaceCan (b) Lift

Figure 1: Illustrations of the *Lift* and *PickPlaceCan* tasks in Robosuite (Zhu et al., 2020b).

We evaluate algorithms in sparse-reward robotic manipulation tasks using the **Robosuite** simulator (Zhu et al., 2020a) and the **Robomimic** dataset suite (Mandlekar et al., 2021). Robosuite is a modular simulation environment supporting both dense and sparse reward configurations. Following the experimental setup of Bhargava et al. (2024), who examined both reward regimes on the *Lift* and *PickPlaceCan* tasks, we focus exclusively on the sparse setting to align with the central objectives of our study. Visual illustrations of the two tasks are provided in Figure 1, and task details are summarized in Table 3.

Robomimic is a standardized benchmark offering human- and machine-generated demonstration datasets. It includes three variants: Proficient Human (PH), Multi-Human (MH), and Machine-Generated (MG). As shown in Bhargava et al. (2024), behavior cloning performs well on PH and MH datasets, while Decision Transformer shows stronger results on the MG variant. Based on these findings, we restrict our experiments to the MG dataset, which contains the lowest-quality demonstrations. This dataset is constructed by sampling rollouts from Soft Actor-Critic (SAC) agents at multiple training checkpoints, thus offering a diverse but suboptimal data distribution.

Table 3: Robosuite Tasks and Dataset Specifications

Attribute	**Lift**	**PickPlaceCan**
Scene Description	A cube is placed on a table in front of a robot arm. The cube's position is randomized each episode.	A bin with four objects is placed in front of the robot, with four nearby target containers. Object positions are randomized per episode.
Objective	Lift the cube vertically above a predefined height.	Pick each object and place it into its corresponding container.
Dataset Composition	244 / 1500 trajectories (300 rollouts × 5 checkpoints), machine-generated via SAC.	716 / 3900 trajectories (300 rollouts × 13 checkpoints), machine-generated via SAC.
Observation & Action Space	Observations: 18D proprioceptive input Actions: 7D Cartesian EE + gripper control	Observations: 22D proprioceptive input Actions: 7D Cartesian EE + gripper control

A.2 D4RL Environments and Datasets

We perform evaluations on continuous control tasks from the **D4RL benchmark suite** (Fu et al., 2020), a widely adopted standard for offline reinforcement learning. Accordingly, we omit detailed elaboration here. Our experiments focus on three locomotion tasks, *Hopper*, *Walker2d*, and *HalfCheetah*, using the *Medium*, *Medium-Replay*, and *Medium-Expert* dataset variants, following the setup in Bhargava et al. (2024).

A.3 Hyperparameters

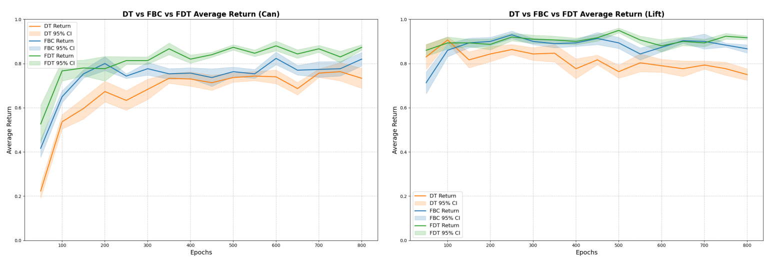
We adopt the hyperparameter setup as Bhargava et al. (2024) and report everything in Table 4 for reproducibility.

Table 4: Hyperparameters for DT, BC, FBC, and CQL evaluations.

Category	Hyperparameter	Value
Transformer (DT)	Number of layers	3
	Attention heads	1
	Embedding dimension	128
	Nonlinearity	ReLU
	Context length	1 (Robomimic), 20 (D4RL)
	Dropout	0.1
	Return-to-go (RTG) conditioning	120 (Robomimic), 6000 (HalfCheetah), 3600 (Hopper), 5000 (Walker)
	Max episode length	1000
MLP (BC)	Network depth	2 layers
	Hidden units per layer	512
	Nonlinearity	ReLU
Training (DT, BC)	Batch size	512 (DT), 100 (BC)
	Learning rate	10^{-4}
	Learning rate decay	0.1 (BC)
	Grad norm clip	0.25
	Weight decay	10^{-4}
	LR scheduler	Linear warmup for first 10^5 steps (DT)
	Epochs	100
Evaluation	Frequency	Every 50 epochs (Robomimic), Every 100 epochs (D4RL)
	Rollouts per eval	50
	Evaluation episodes	100
	Seeds	5
	Reference (DT)	https://github.com/kzl/decision-transformer
CQL	Batch size	2048
	Steps per iteration	1250
	Iterations	100
	Discount factor	0.99
	Policy learning rate	3×10^{-4}
	Q-function learning rate	3×10^{-4} (D4RL), 1×10^{-3} (Robomimic)
	Actor MLP dimensions	[300, 400] (Robomimic)
	Soft target update rate	5×10^{-3}
	Target update period	1
	Alpha multiplier	1
	CQL n_actions	10
	Min Q weight	5
	CQL temperature	1
	Importance sampling	True
	Lagrange tuning	False
	Target action gap	-1
	Lagrange threshold τ	5 (Robomimic)
	Reference (CQL)	https://github.com/tinkoff-ai/CORL

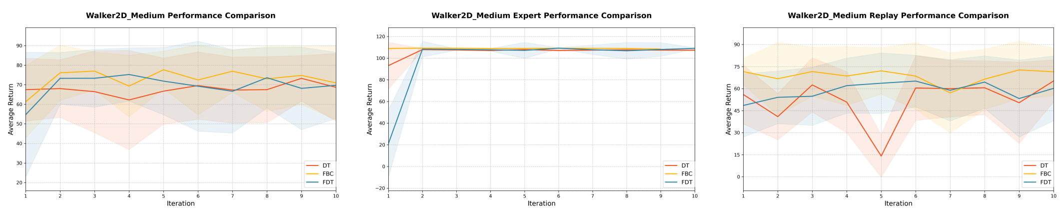
B Learning Curves

This section presents the full learning curves for all evaluated methods. Figure 2 shows training performance on the RoboMimic sparse tasks (*Lift* and *PickPlaceCan*), with 95% confidence intervals. Figure 3 provides learning curves for the D4RL locomotion benchmarks across all nine task-dataset combinations. CQL curves are excluded for clarity, as comprehensive results are already provided in Bhargava et al. (2024) and fall outside the core comparative scope of this study.

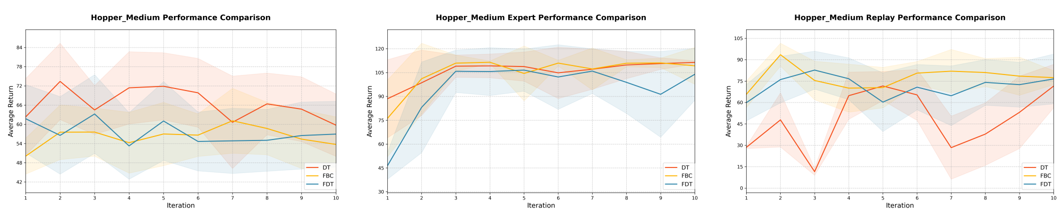


(a) Results for the task PickupCan (b) Results for the task Lift

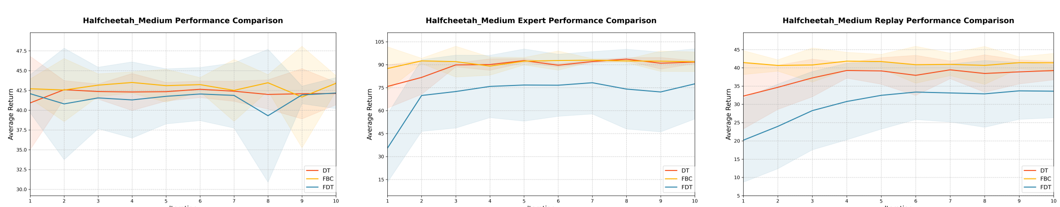
Figure 2: Results for Robomimic tasks. Performance comparison of FBC, FDT, DT.



(a) Walker2d (Medium) (b) Walker2d (Medium-Expert) (c) Walker2d (Medium-Replay)



(d) Hopper (Medium) (e) Hopper (Medium-Expert) (f) Hopper (Medium-Replay)



(g) Halfcheetah (Medium) (h) Halfcheetah (Medium-Expert) (i) Halfcheetah (Medium-Replay)

Figure 3: Performance of FBC, FDT, and DT on D4RL.

References

D. Bhargava, S. Paternain, M. R. Rilo, L. Pinto, P. Abbeel, and A. Filos. When should we prefer decision transformers for offline reinforcement learning? In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Yevgen Chebotar, Quan Vuong, Karol Hausman, Fei Xia, Yao Lu, Alex Irpan, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, et al. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions. In *Conference on Robot Learning*, 2023.

L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Scott Emmons, Zichuan Wang, Aditya Anand, Evan Zheran Langlois, Yikai Tian, Xuechen Li, Yilun Du, Ahmed Touati, Sudeep Choudhury, Seyed Kamyar Seyed Ghasemipour, et al. RvS: What is essential for offline rl via supervised learning? In *NeurIPS Offline Reinforcement Learning Workshop*, 2021.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Hiroki Furuta, Yutaka Matsuo, and Shixiang Shane Gu. Generalized decision transformer for offline hindsight information matching. In *International Conference on Learning Representations*, 2022.

Pierre-Louis Guhur, Shizhe Chen, Ricardo Garcia Pinel, Makarand Tapaswi, Ivan Laptev, and Cordelia Schmid. Instruction-driven history-aware policies for robotic manipulations. In *Conference on Robot Learning*, 2023.

Shengchao Hu, Li Shen, Ya Zhang, and Dacheng Tao. Graph decision transformer. *arXiv preprint arXiv:2303.03747*, 2023.

Shengchao Hu, Ziqing Fan, Chaoqin Huang, Li Shen, Ya Zhang, Yanfeng Wang, and Dacheng Tao. Q-value regularized transformer for offline reinforcement learning. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

M. Janner, Q. Li, S. Levine, and C. Finn. Trajectory transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative Q-learning for offline reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H.-T. Lin (eds.), *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 1179–1191. Curran Associates, Inc., 2020.

Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, et al. Pre-trained language models for interactive decision-making. *Advances in Neural Information Processing Systems*, 2022.

Wenzhe Li, Hao Luo, Zichuan Lin, Chongjie Zhang, Zongqing Lu, and Deheng Ye. A survey on transformers in reinforcement learning. *Transactions on Machine Learning Research*, 2023.

Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín Wang, Li Fei-Fei, Silvio Savarese, Yuke Zhu, et al. What matters in learning from offline human demonstrations for robot manipulation. *Robotics: Science and Systems (RSS)*, 2021.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, 2020.

Keiran Paster, Sheila McIlraith, and Jimmy Ba. You can't count on luck: Why decision transformers and rvs fail in stochastic environments. *Advances in neural information processing systems*, 2022.

N. M. M. Shafiullah, Z. An, Q. Luo, M. Janner, T. Jaakkola, B. Boots, and S. Levine. Behavior transformers: Cloning k modes with one stone. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Yujin Tang and David Ha. The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning. *Advances in Neural Information Processing Systems*, 2021.

Denis Tarasov, Alexander Nikulin, Dmitry Akimov, Vladislav Kurenkov, and Sergey Kolesnikov. CORL: Research-oriented deep offline reinforcement learning library. *Advances in Neural Information Processing Systems*, 2023.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30 (NIPS)*, pp. 5998–6008. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Kerong Wang, Hanye Zhao, Xufang Luo, Kan Ren, Weinan Zhang, and Dongsheng Li. Bootstrapped transformer for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 2022.

Taku Yamagata, Ahmed Khalil, and Raul Santos-Rodriguez. Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline rl. In *International Conference on Machine Learning*, 2023.

Yuke Zhu, Ajay Mandlekar, Animesh Gao, Li Fei-Fei, Silvio Savarese, et al. Robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020a.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Kevin Lin, Abhiram Maddukuri, Soroush Nasiriany, and Yifeng Zhu. Robosuite environments documentation. https://robosuite.ai/docs/modules/environments.html, 2020b.