Bootstrapped Transformer for Offline Reinforcement Learning - Appendix

Kerong Wang
Shanghai Jiao Tong University
wangkerong@sjtu.edu.cn

Hanye Zhao
Shanghai Jiao Tong University
fineartz@sjtu.edu.cn

Xufang Luo
Microsoft Research Asia
xufluo@microsoft.com

Kan Ren✉
Microsoft Research Asia
kan.ren@microsoft.com

Weinan Zhang
Shanghai Jiao Tong University
wnzhang@sjtu.edu.cn

Dongsheng Li
Microsoft Research Asia
dongsli@microsoft.com

A Training Details

In this part, we list the details of our experiments, including the hyperparameter setting and the resources used to train the model. We also provide the source code of our paper in the supplementary materials.

A.1 Hyperparameter Setting

Table 1: Hyperparameters and search range of parameter-tuning for experiments.

Hyperparameters	Search Range
Generation Threshold k	$\{0.0, 0.2, 0.4, 0.6, 0.8\}$
Generation Percentage $\eta\%$ (BooT-o)	$\{2\%, 4\%, 6\%, 8\%, 10\%\}$
Generation Percentage $\eta\%$ (BooT-r)	$\{0.2\%, 0.4\%, 0.6\%, 0.8\%, 1.0\%\}$
Generation Length T'	$\{1, 3, 5, 9\}$
Initial Learning Rate λ	$\{1 \times 10^{-3}, 1 \times 10^{-4}\}$
Training Epoch Number E (Adroit Only)	$\{5, 10, 15, 20\}$
Planning Horizon H (Adroit Only)	$\{5, 10, 15\}$

In this section, we list all hyperparameters we searched in our experiments corresponding to Sec. 5 in Table 1. For fair comparison, we do not change most of the default hyperparameters of TT [2] and we do not list them in the table. We also use Adam optimizer combined with linear warmup and cosine decay scheduling on the learning rate as the same as TT.

For experiments of adding noise to data, corresponding to TT+S4RL setting in Sec. 5.2 and Sec. 5.3 we use zero-mean Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma I), \sigma = 3 \times 10^{-4}$, following the same setting in S4RL [3]. We first add noise to the normalized original data, then discretize the noisy data using the same discretizer as the original dataset.

A.2 Training Resources

We use one NVIDIA Tesla V100 GPU to train each model. Training with teacher-forcing generation typically requires 10-16 hours, which depends on different dataset, and training with autoregressive generation typically requires 20-72 hours. Note that, the utilized training samples are the same for all the compared methods to keep fair comparison.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

B Licences

The D4RL [1] dataset we use is licensed under the Creative Commons Attribution 4.0 License (CC BY), which can be found in

`https://github.com/rail-berkeley/d4rl/blob/master/README.md`.

We also use the code of TT [2], which uses MIT Licence as in

`https://github.com/jannerm/trajectory-transformer/blob/master/LICENSE`.

C Additional Experiments

C.1 Experiments on Random and Expert Dataset

We perform additional experiments on both the random and the expert dataset, and the results are listed in Table 2. From the results, we cannot see significant differences on these datasets. This is as expected since expert datasets are usually used to test imitation learning algorithms, but not offline RL algorithms; and the quality of random datasets is too poor, which makes the performance quite similar.

Table 2: Experiment results on random and expert dataset. We report the mean results corresponding to 15 random seeds (5 training seeds for independently trained Transformers and 3 evaluation seeds for each model).

Dataset	**Environment**	**TT**	**BooT-o, AR**	**BooT-r, AR**	**BooT-o, TF**	**BooT-r, TF**
Expert	HalfCheetah	95.3	92.3	94.4	95.0	**95.4**
Expert	Hopper	102.3	110.3	**110.5**	104.6	108.2
Expert	Walker2D	108.4	108.5	**108.7**	108.5	108.5
Average		102.0	103.7	**104.6**	102.7	104.1
Random	HalfCheetah	7.9	6.7	6.9	4.6	7.5
Random	Hopper	6.7	6.8	6.5	6.5	6.6
Random	Walker2D	5.6	5.2	4.8	4.6	4.8
Average		6.8	6.3	6.1	5.2	6.3

C.2 Experiments on Random-mixed Dataset

We also perform experiments on random-mixed dataset, where we replace $\phi\%$ of the medium-replay dataset with the random dataset, to test the impact of less-performing data on our algorithm. The experiments are performed on HalfCheetah, Hopper, and Walker2d environments and the results are listed in Table 3. The results demonstrate that, in general, the performance of all methods decreases as the ratio of random data increases. Among all the methods, BooT-r, AR achieves the best results until using a pure random dataset, when the performance of all methods drops to the same level.

Table 3: Experiment results on medium-replay dataset with $\phi\%$ replaced by random dataset. We report the mean results corresponding to 15 random seeds.

$\phi\%$	**TT**	**BooT-o, AR**	**BooT-r, AR**	**BooT-o, TF**	**BooT-r, TF**
20%	57.2	50.8	**66.0**	51.0	55.9
40%	53.7	47.6	**61.6**	39.6	39.0
60%	32.5	25.7	**44.8**	17.6	28.0
80%	11.1	10.2	12.8	**13.8**	11.5
Pure Random	**6.8**	6.3	6.1	5.2	6.3

C.3 Training CQL Agent with Generated Data

One natural question raises that, are the generated data useful for other algorithms? In this section, we try to train CQL agent with additional data generated by BooT and present the results in Table 4. We can see that CQL does not perform well with generated data from BooT. The reason mainly lies

Table 4: Results on CQL using additional data generated by BooT on adroit domain.

Dataset	Environment	BooT + CQL	BooT	CQL
Med-Expert	HalfCheetah	5.0	94.0	91.6
Med-Expert	Hopper	0.8	102.3	105.4
Med-Expert	Walker2d	26.4	110.4	108.8
Medium	HalfCheetah	30.0	50.6	44.0
Medium	Hopper	79.8	70.2	58.5
Medium	Walker2d	6.4	82.9	72.5
Med-Replay	HalfCheetah	4.3	46.5	45.5
Med-Replay	Hopper	5.0	92.9	95.0
Med-Replay	Walker2d	5.8	87.6	77.2
Average		18.2	81.9	77.6

in two aspects. First, it is non-trivial to utilize the generated data from BooT to train CQL agent. Specifically, since generated data are discretized, while CQL uses original continuous data format from the environment, we need to recover the input data from discrete tokens, which may cause information loss. Our BooT does not require such recovering operation, because it directly models the distribution of discretized data. Second, BooT is a self-improving method, which uses the generated data to further improve the sequence model itself. Although it is feasible to use generated data from BooT to train other models, like what we have done here, such utilization is not consistent with the *self-improving* idea in BooT, and makes BooT more like a generative model which requires further design. Though it is beyond the scope of our work, we believe it is a promising direction and leave it as future work.

C.4 Numerical Results of TT Retrained with Loss-weighted Data

This section provides the numerical results of TT retrained with loss-weighted data in Table 5, corresponding to Figure 6 in Sec. 5.5. We choose the data with the lowest / largest $\eta\%$ loss in each batch to re-train the model, and report the mean results corresponding to 15 random seeds.

Table 5: TT-retrain results with data of the largest / lowest $\eta\%$ loss in each batch, corresponding to Largest / Lowest First in the header row.

	TT (Re-train, Loss-weighted)			
$\eta\%$	**Largest First**	**Lowest First**	**Baselines**	
2%	69.5	71.4	**TT (Original)**	72.6
4%	66.9	71.8	**TT (Reproduce)**	70.1
6%	66.8	67.4	**TT (Retrain)**	69.4
8%	70.5	71.9	**BooT**	81.9
10%	66.0	69.2		

D Detailed Settings of Experiments

D.1 Distance Calculation

In this section, we introduce the details of distance calculation in Sec. 5.3. We calculate the distances from the original trajectories set $X = \left\{\boldsymbol{\tau}_{(i)}\right\}_{i=1}^{N}$ to the corresponding generated trajectories set $\tilde{X} = \left\{\tilde{\boldsymbol{\tau}}_{(i)}\right\}_{i=1}^{N}$, denoted as $d(X, \tilde{X})$, to analyze the numerical characteristics of generated trajectories. In Table 4, we use two different distance metrics: *RMSE* standing for Root Mean Squared Error, and *MMD* standing for the Maximum Mean Discrepancy. We also calculate the distances between the generated trajectories and the original trajectories in both training and evaluation stages.

For RMSE, the distance is calculated as

$$d_{\text{RMSE}}(X, \tilde{X}) = \frac{1}{N} \sum_{i=1}^{N} \left\| \boldsymbol{\tau}_{(i)} - \tilde{\boldsymbol{\tau}}_{(i)} \right\|_2 . \tag{1}$$

And for MMD, the distance is calculated using Gaussian kernel as

$$
\begin{aligned}
k(\boldsymbol{\tau}_{(i)}, \tilde{\boldsymbol{\tau}}_{(i)}) &= \exp\left(\frac{-\left\|\boldsymbol{\tau}_{(i)} - \tilde{\boldsymbol{\tau}}_{(i)}\right\|^2}{2\sigma^2}\right) \\
d^2_{\text{MMD}}(X, \tilde{X}) &= \frac{\sum_{i,j\neq i} k(\boldsymbol{\tau}_{(i)}, \boldsymbol{\tau}_{(j)})}{N(N-1)} - \frac{2\sum_{i,j} k(\boldsymbol{\tau}_{(i)}, \tilde{\boldsymbol{\tau}}_{(j)})}{N^2} + \frac{\sum_{i,j\neq i} k(\tilde{\boldsymbol{\tau}}_{(i)}, \tilde{\boldsymbol{\tau}}_{(j)})}{N(N-1)} .
\end{aligned} \tag{2}
$$

We also calculate the RMSE with discretized trajectories since we are using discretized trajectories to train our model. As for the MMD metric, we use continuous trajectories because Gaussian kernel is hardly applied to discrete data. We reconstruct the discrete trajectories into continuous ones through replacing each discrete value by the middle value of its bin.

In the training stage, we calculate the distances from the generated trajectories to their corresponding original offline trajectories, denoted as *Dataset* in Table 4. Recall that we only generate the last T' timesteps of new trajectories in Sec. 4.1 for BooT, we extract the last T' timesteps of the trajectories and calculate the distance on them.

Additionally, we also calculate the distances from the state in generated trajectories to the state in real trajectories from the environment in the evaluation stage, denoted as *Environment* in Table 4. For both Dataset and Environment with TT + S4RL setting, we add random noise as described in Appendix A.1, and calculate the distance of the last T' timesteps as same as BooT.

In the evaluation phase, the inter-state distance is a reasonable choice to estimate the difference between the learned transition dynamics and the real transition probability, as it is essentially calculating

$$
\text{Distance} = \left\|\hat{s}_{t+1} - s_{t+1}\right\|^2, \quad \hat{s}_{t+1} \sim \tilde{P}(s'|s = s_t, a = a_t),\ s_{t+1} \sim P(s'|s = s_t, a = a_t), \tag{3}
$$

where $\tilde{P}(s'|s, a)$ is the learned model and $P(s'|s, a)$ is the ground-truth transition probability. From the formula, we can see that the inter-state distance gives an reasonable estimation of the error between the learned transition dynamics and the real ones.

D.2 Visualization

In this section, we provide the complete visualization results in Sec. 5.3 as Figure 1 to 18. They contain visualized data distributions of original offline trajectories, trajectories with teacher-forcing generation, and trajectories with autoregressive generation. We randomly sample 2500 trajectories from the original dataset and save their corresponding generated trajectories in the last training epoch. We then excerpt the last T' timesteps of them because we only generate these timesteps for each original trajectory, as described in Sec. 4.1. Finally we reduce all data into 2-dimension together using t-SNE [4] algorithm. It is clearly illustrated in most figures that a large portion of data generated by the teacher-forcing method overlaps with the support of the original dataset, while those generated autoregressively lie out of the original data distribution, demonstrating the ability of BooT to expand the data coverage. This finding is aligned to the discussion in our main paper.

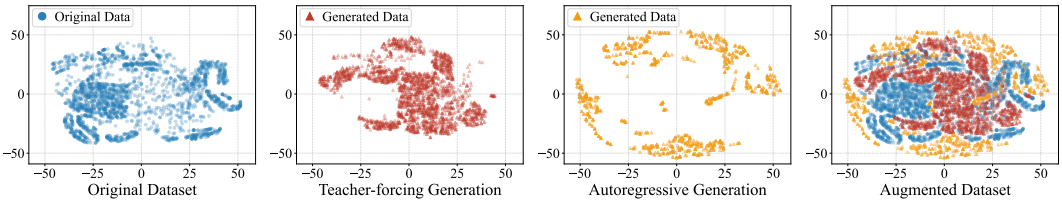

Figure 1: Distribution of last T' timesteps of the trajectories from original offline dataset, trajectories with teacher-forcing generation, and trajectories with autoregressive generation in halfcheetah-medium-expert task.

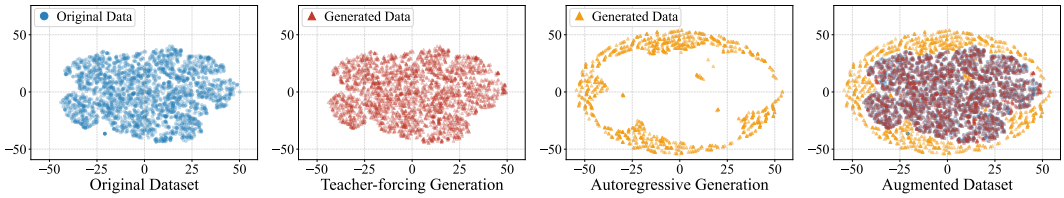

Figure 2: Distribution of last T' timesteps of the trajectories without reward and reward-to-go in halfcheetah-medium-expert task.

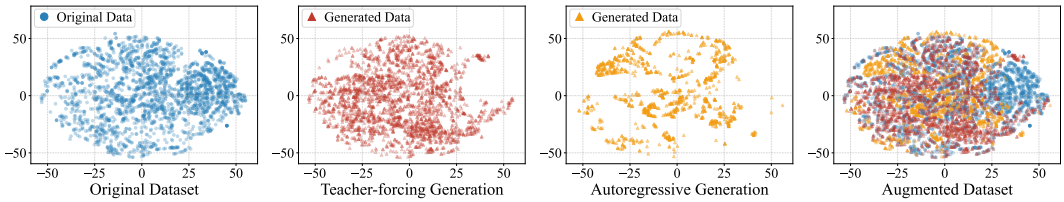

Figure 3: Distribution in hopper-medium-expert task.

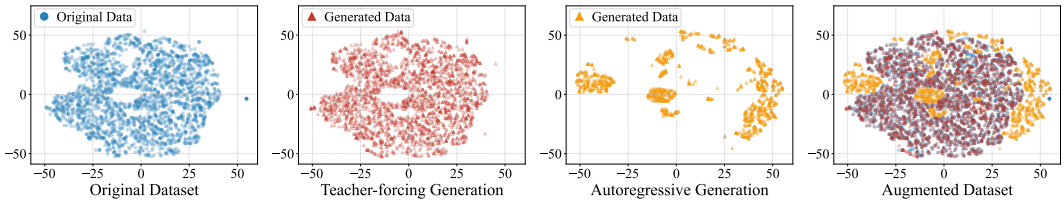

Figure 4: Distribution without reward and reward-to-go in hopper-medium-expert task.

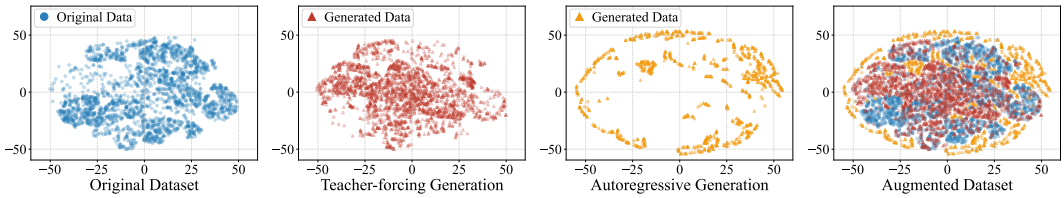

Figure 5: Distribution in walker2d-medium-expert task.

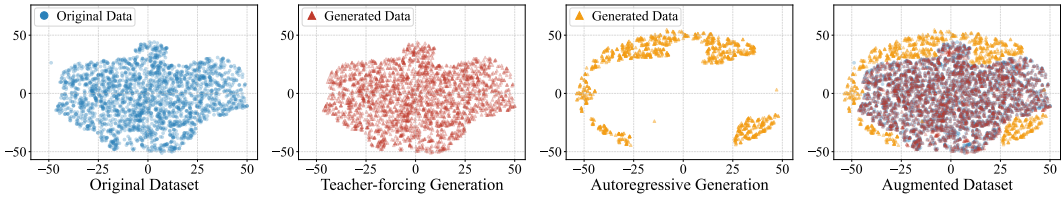

Figure 6: Distribution without reward and reward-to-go in walker2d-medium-expert task.

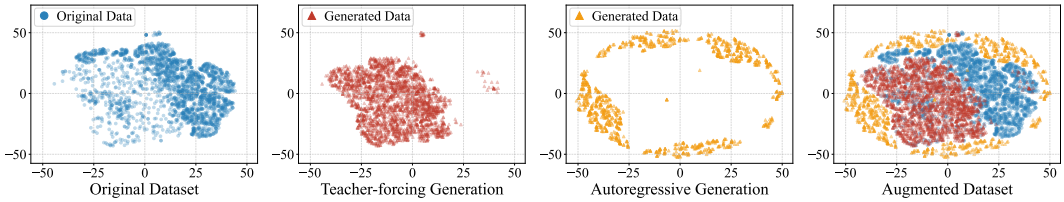

Figure 7: Distribution in halfcheetah-medium task.

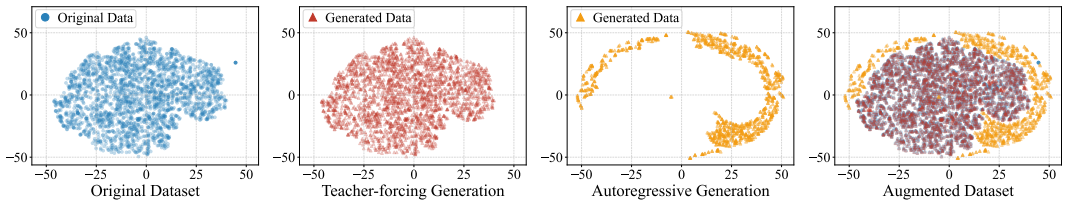

Figure 8: Distribution without reward and reward-to-go in halfcheetah-medium task.

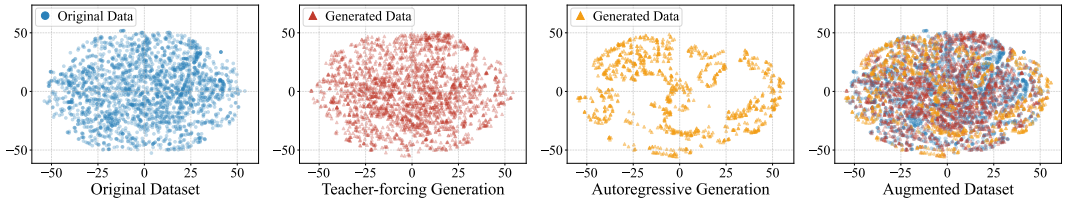

Figure 9: Distribution in hopper-medium task.

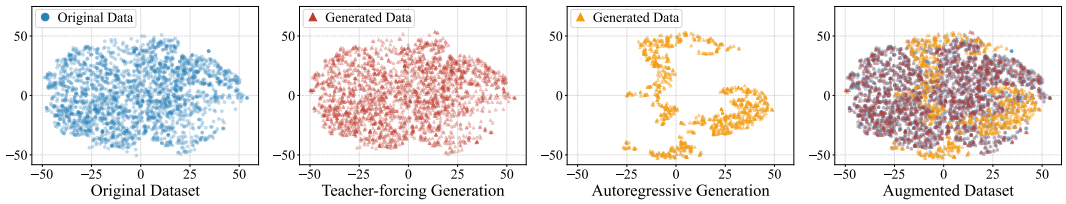

Figure 10: Distribution without reward and reward-to-go in hopper-medium task.

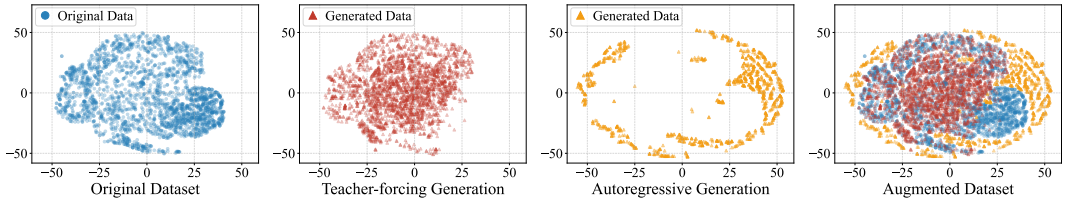

Figure 11: Distribution in walker2d-medium task.

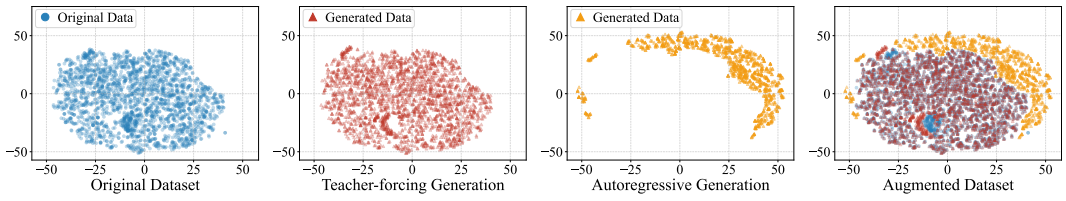

Figure 12: Distribution without reward and reward-to-go in walker2d-medium task.

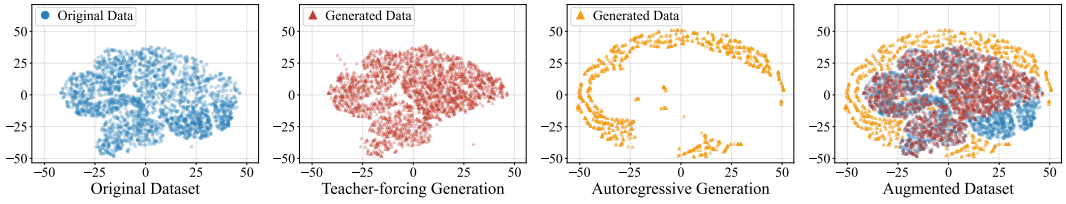

Figure 13: Distribution in halfcheetah-medium-replay task.

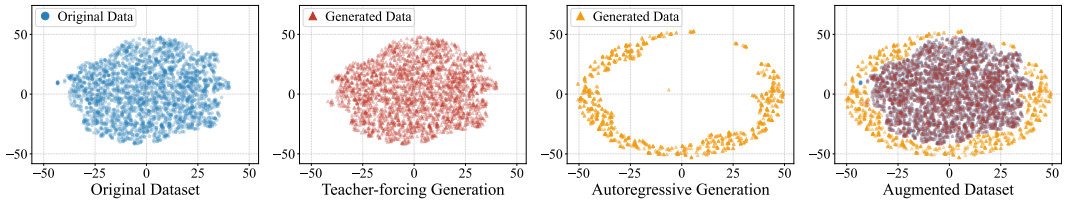

Figure 14: Distribution without reward and reward-to-go in halfcheetah-medium-replay task.

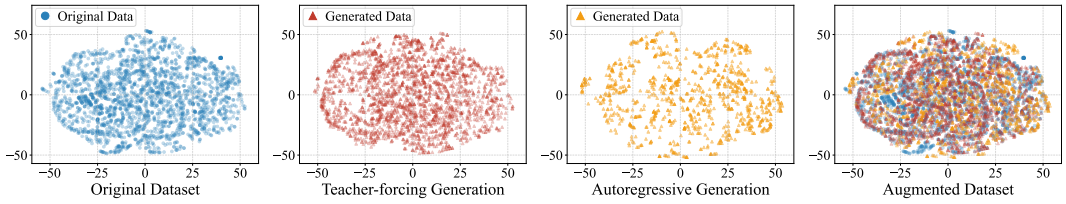

Figure 15: Distribution in hopper-medium-replay task.

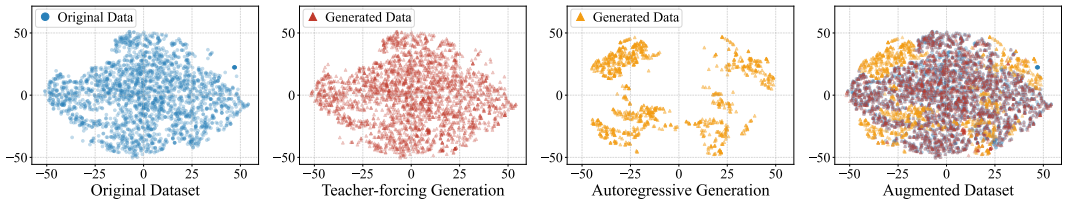

Figure 16: Distribution without reward and reward-to-go in hopper-medium-replay task.

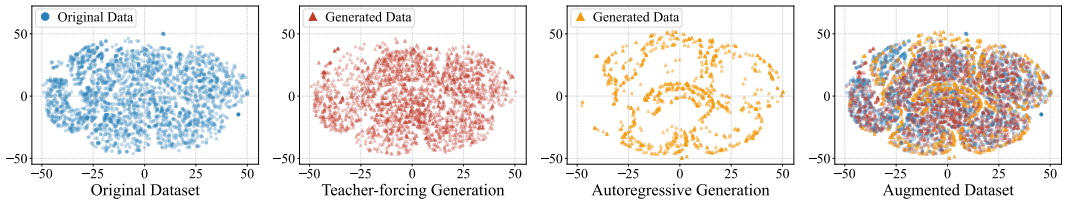

Figure 17: Distribution in walker2d-medium-replay task.

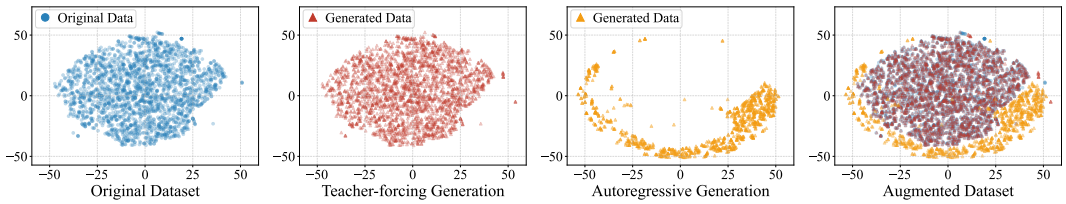

Figure 18: Distribution without reward and reward-to-go in walker2d-medium-replay task.

References

[1] 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.

[2] Janner Michael, Li Qiyang, and Levine Sergey. Offline reinforcement learning as one big sequence modeling problem. In *NeurIPS*, 2021.

[3] Samarth Sinha, Ajay Mandlekar, and Animesh Garg. S4RL: surprisingly simple self-supervision for offline reinforcement learning in robotics. In *CoRL*, 2021.

[4] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 2008.