Offline RL Literature

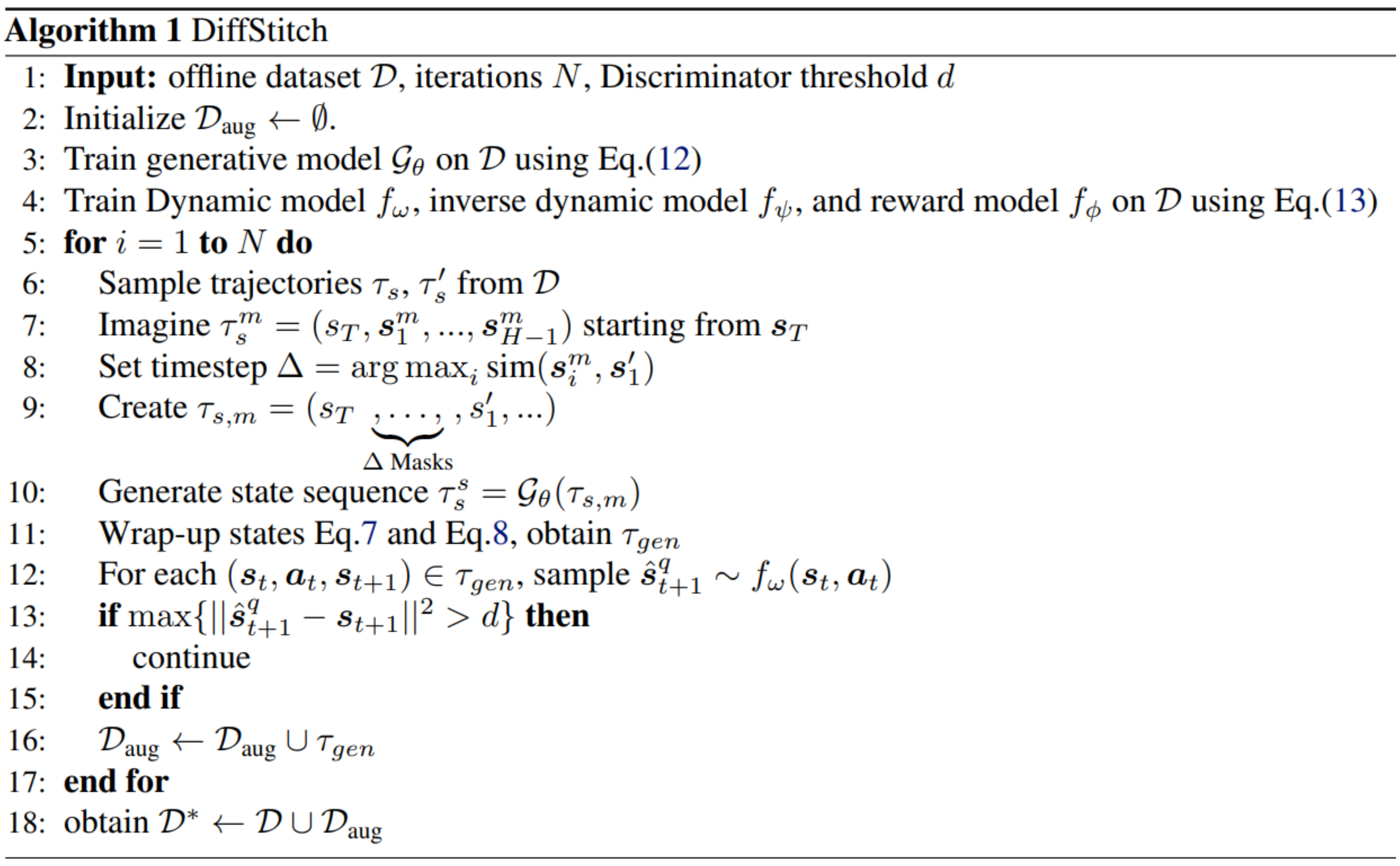
DiffStitch: Boosting Offline Reinforcement Learning with Diffusion-based Trajectory Stitching

Guanghe Li [1] Yixiang Shan [1] Zhengbang Zhu [2] Ting Long [1] Weinan Zhang [2]

About

arXiv link: https://arxiv.org/abs/2402.02439

algorithm schematic:



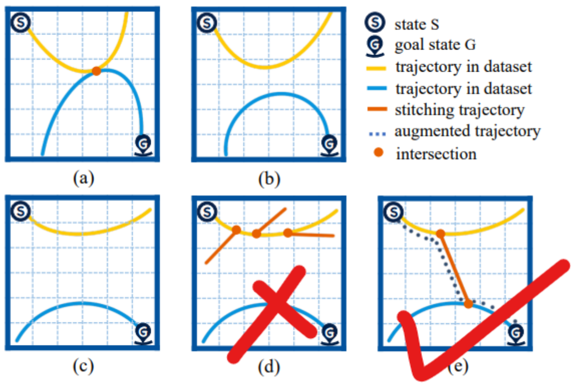
Algorithm 1 DiffStitch

```
1: Input: offline dataset $\mathcal{D}$, iterations $N$, Discriminator threshold $d$
2: Initialize $\mathcal{D}_{\text{aug}} \leftarrow \emptyset$.
3: Train generative model $\mathcal{G}_\theta$ on $\mathcal{D}$ using Eq.(12)
4: Train Dynamic model $f_\omega$, inverse dynamic model $f_\psi$, and reward model $f_\phi$ on $\mathcal{D}$ using Eq.(13)
5: for $i = 1$ to $N$ do
6:     Sample trajectories $\tau_s$, $\tau'_s$ from $\mathcal{D}$
7:     Imagine $\tau_s^m = (s_T, \boldsymbol{s}_1^m, ..., \boldsymbol{s}_{H-1}^m)$ starting from $\boldsymbol{s}_T$
8:     Set timestep $\Delta = \arg\max_i \text{sim}(\boldsymbol{s}_i^m, \boldsymbol{s}'_1)$
9:     Create $\tau_{s,m} = (s_T \underbrace{, \dots ,}_{\Delta \text{ Masks}} , s'_1, ...)$
10:    Generate state sequence $\tau_s^s = \mathcal{G}_\theta(\tau_{s,m})$
11:    Wrap-up states Eq.7 and Eq.8, obtain $\tau_{gen}$
12:    For each $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \in \tau_{gen}$, sample $\hat{\boldsymbol{s}}_{t+1}^q \sim f_\omega(\boldsymbol{s}_t, \boldsymbol{a}_t)$
13:    if $\max\{||\hat{\boldsymbol{s}}_{t+1}^q - \boldsymbol{s}_{t+1}||^2 > d\}$ then
14:        continue
15:    end if
16:    $\mathcal{D}_{\text{aug}} \leftarrow \mathcal{D}_{\text{aug}} \cup \tau_{gen}$
17: end for
18: obtain $\mathcal{D}^* \leftarrow \mathcal{D} \cup \mathcal{D}_{\text{aug}}$
```

2024, Feb 2, SJTU, ICML in submission

Intro

1. augmented data should reach the reward-dense states
2. link reward-sparse with reward-dense



> Intuitively, if we augment the offline dataset by stitching together low reward trajectories with high-reward trajectories, the policy may learn the ability to transit from low-reward states to high-reward states and ultimately achieve a higher overall reward

DiffStitch is the first offline RL **augmentation method** that generates sub-trajectories to stitch any two trajectories.

1. **randomly selects a low-reward trajectory and a high-reward trajectory**
2. **generates a sub-trajectory to stitch them together**.
3. Through our paradigm, one can easily **transfer the trajectories trapped in low rewards to the one with high reward**s, enhancing the learning of policy

---

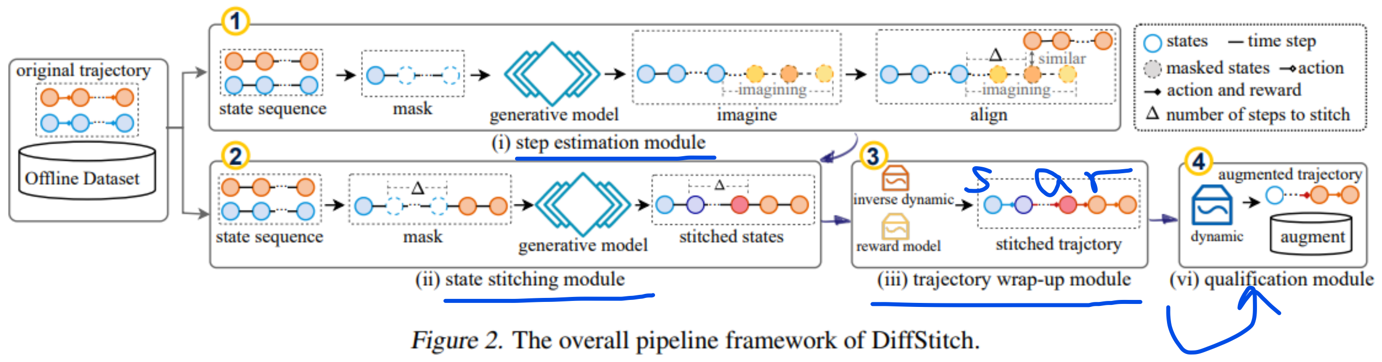


Figure 2. The overall pipeline framework of DiffStitch.

Pipeline details

Four modules

1. The **step estimation module** estimates the number of time steps required for stitching two trajectories

> Given the states of two trajectory $\tau_s = (s_1, s_2 \dots s_T)$ and $\tau_s' = (s_1', s_2', \dots, s_{T'}')$, estimate the number of the step required for state s_T to **transit naturally** to state s_1'. Here, s_T could be viewed as the start state and s_1' could view as the target state for stitching.
>
> $s_T \to \cdot \to \cdot \to \dots \to x \to s_1'.$
>
> Suppose s_T can reach state x in exactly i steps. s. The higher the similarity between s_1' and x, the more probable it is for sT to reach s ' 1 in exactly i steps as well.
>
> 1. Generate a length H trajectory **conditioned** on the first state being s_T:
>
> $$\begin{aligned}\tau_s^m &= (s_T, s_1^m, \dots, s_{H-1}^m) \\ &= \mathcal{G}_\theta((s_T, [\text{MASK}], \dots, [\text{MASK}]))\end{aligned}$$
>
> 2. Measure the similarity of the each state of the generated Markov Chain with the terminal state s_1'

$$\Delta = \arg\max_i \operatorname{sim}\left(s_i^m, s_1'\right)$$

We implement sim with cosine similarity.

Implementing the diffusion model $\mathcal{G}_\theta$

a. Training

Pick length H state trajectories $\tau_s = (s_1, s_2, \ldots, s_H)$ from the offline dataset.

Reveal the initial state of each sample s_i^1 (make sure the Diffusion model is a conditional generator)

Randomly mask some of the rest of the states, forming masked training data. We ensure the last state is masked, so that we will be able to compare and select the best generated trajectory by measure the similarity score between the generated s_i^m and s_1'.

$$\tau_{s,m} = (s_1, \ldots, s_i, [\text{MASK}], [\text{MASK}], \ldots, s_t, s_{+1}, [\text{MASK}])$$

$d_\mu(\pi) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t p(S_t = s \mid S_0 = s_0)$

μ

Next, we randomly sample a noise $\epsilon \sim \mathcal{N}(0, \mathcal{I})$ and a diffusion timestep $k \sim \mathcal{U}\{1, \ldots, K\}$, and train $\mathcal{G}_\theta$ with:

$$\mathcal{L}_\theta = \mathbb{E}_{k, \tau_s \in \mathcal{D}} \left\| \epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}}\tau_s + \sqrt{1-\bar{\alpha}}\epsilon, \tau_{s,m}, k\right) \right\|^2.$$

We implement the noise prediction network ϵ_θ with a U-net, because it excels at few-shot learning since our offline dataset is assumed to be small

“U-Net is designed to **work well with very few training samples**. This is particularly advantageous in medical applications where annotated images can be scarce and expensive to obtain.”

$P(s' \mid a, s)\,\pi_\beta(a \mid s)$

$d_\mu(\cdot) =$

b. Inference

Denoise for K steps. For noise-masked state sequence $\tau_{s,m}$, the diffusion model $\mathcal{G}_\theta$ reconstruct the sample with K denoising steps :

$R \leftarrow \frac{(s, a, s')}{\text{agent}}$ $d_\mu(s, \pi)$ $P(\cdot \mid s)$ $\pi_k(a \mid s)$

$$\tau_{s,m}^{k-1} = \frac{1}{\sqrt{\alpha_k}}\left(\tau_{s,m}^k - \frac{1-\alpha_k}{\sqrt{(1-\bar{\alpha}_k)}}\epsilon_\theta\left(\tau_{s,m}^k, \tau_{s,m}, k\right)\right) + \sqrt{1-\alpha_k}\epsilon,$$
$$\epsilon \sim \mathcal{N}(0, \mathbf{I}), \text{ for } k = K, \ldots, 1$$

where α_i is the cosine function of k and $\bar{\alpha}_i = \prod_1^i \alpha_i \cdot \mathcal{N}$

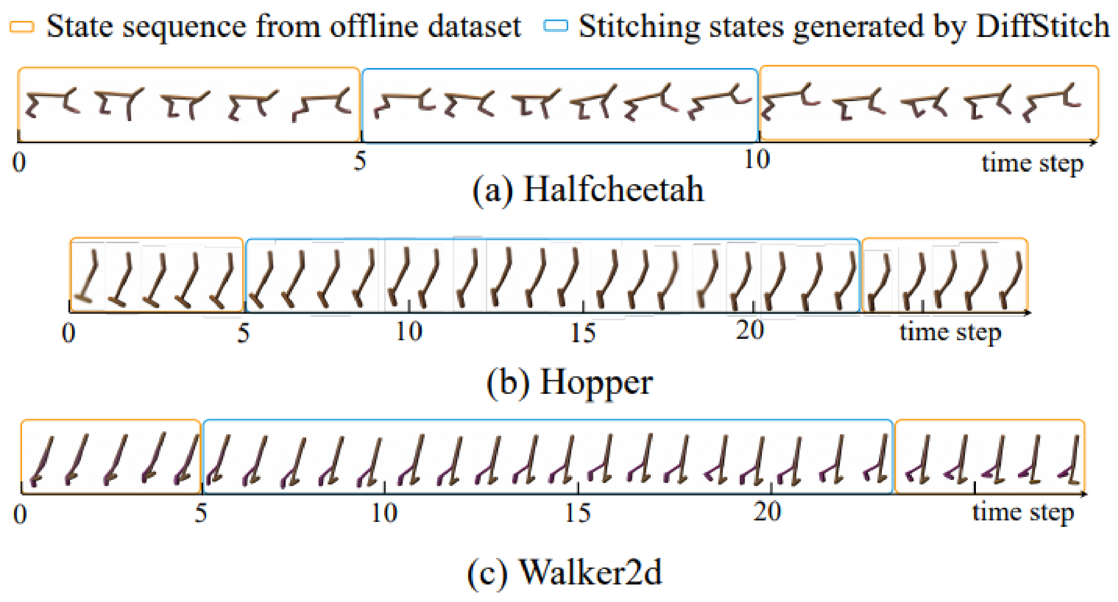
π_k

2. the estimated number of time steps is input into the **state stitching module**, which generates states consistent with the number of estimated steps.

Generate trajectory with length Δ so that the the augmented trajectory fragment reaches the nearest point to s_1'.

Then append the generated trajectory with the first few states in the second trajectory τ', forming a synthetic mixture trajectory of length H:

$$\begin{aligned} \tau_s^s &= \mathcal{G}_\theta\left(\tau_{s,m}\right) \\ &= (s_T, \underbrace{\widetilde{s_1}, \ldots, \widetilde{s_\Delta}}_{\tilde{\tau}_s:\ \Delta \text{ stitching states}}, s_1', \ldots, s_{H-1-\Delta}') \end{aligned}$$



3. Next, **the trajectory wrap-up module** predicts the rewards and actions based on the generated states, to obtain full synthesized trajectories.

a. First we train an inverse dynamic model f_ψ on the offline dataset

$$\mathcal{L}_\phi = \mathbb{E}_{(\boldsymbol{s}_t,\boldsymbol{a}_t,r_t,\boldsymbol{s}_{t+1})\mathcal{D}}\left[\boldsymbol{a}_t - f_\phi\left(\boldsymbol{s}_t, \boldsymbol{s}_{t+1}\right)\right]$$

f_ψ is implemented as an MLP.

Then we decode the actions in between the generated states:

$$\begin{aligned}&\text{Input } \left(\hat{s}_t, \hat{s}_{t+1}\right) \in \left\{(\boldsymbol{s}_T, \widetilde{s_1}), (\widetilde{s_1}, \widetilde{s_2}), \ldots, \left(\widetilde{s_\Delta}, s_1'\right)\right\}\\ &\text{Output } \widetilde{\boldsymbol{a}_t} = f_\psi\left(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{s}}_{t+1}\right)\end{aligned}$$

b. We also train a reward model on the offline data that predicts the reward function of the MDP:

$$\mathcal{L}(\phi) = \mathbb{E}_{(\boldsymbol{s}_t,\boldsymbol{a}_t,r_t,\boldsymbol{s}_{t+1})\sim\mathcal{D}}\left[r_t - f_\phi\left(\boldsymbol{s}_t, \boldsymbol{a}_t\right)\right]$$

f_ϕ is also implemented by an MLP.

Based on the states and actions, we use a reward model f_ϕ to deduce the rewards obtained from the generated trajectory.

$$\widetilde{r_t} = f_\phi\left(\hat{\boldsymbol{s}}_t, \widetilde{\boldsymbol{a}_t}\right)$$

c. Inserting the decoded actions and estimated rewards into the generated state trajectory, we form the generated full trajectory:

$$\tau_r = \left\{(\boldsymbol{s}_T, \widetilde{\boldsymbol{a}_T}, \widetilde{r_T}), (\widetilde{\boldsymbol{s}_1}, \widetilde{\boldsymbol{a}_1}, \widetilde{r_1}), \ldots, (\widetilde{\boldsymbol{s}_\Delta}, \widetilde{\boldsymbol{a}_\Delta}, \widetilde{r_\Delta}), \left(\boldsymbol{s}_1' \, \boldsymbol{a}_1', r_1'\right)\right\}$$

We denote τ_r as the stitching trajectory.

4. Finally, the **qualification module** evaluates the quality of the new trajectory. If qualified, augments the dataset with the new trajectory.

We assess the quality of generated trajectories, **filtering out low-quality data** while retaining high-quality data to **ensure** that the stitching trajectories **align with environmental dynamics**

Model-based method:

First train a dynamics model $f_\omega : \mathcal{S} \times \mathcal{S} \to \mathcal{A}$ using the offline dataset that approximates $P(\cdot|s, a)$ of the MDP. The model is implemented by an MLP:

$$\mathcal{L}(\omega) = \widehat{\mathbb{E}}_{s_t, a_t, s_{t+1} \sim \mathcal{D}_{\text{offline}}} \left[(s_{t+1} - f_\omega(s_t, a_t))^2 \right]$$

then we evaluate the generated state according to whether it deviates significantly from the dynamic model's estimate:

$$\left\| \hat{s}_{t+1} - f_\omega\left(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{a}}_t\right) \right\|^2 \geq \delta$$

Can we directly plug the dynamic estimator into the DDPM sampler to save time

Discarding bad generations wastes alot. Kind of making it self-supervised.

By employing the pipeline above, we can obtain a new augmented trajectory

$\tau_{\text{gen}} = \{(\boldsymbol{s}_1, \boldsymbol{a}_1, r_1), \ldots, (\boldsymbol{s}_T, \widetilde{\boldsymbol{a}_T}, \widetilde{r_T}), (\widetilde{\boldsymbol{s}_1}, \widetilde{\boldsymbol{a}_1}, \widetilde{r}_1), \ldots, (\widetilde{\boldsymbol{s}_\Delta}, \widetilde{\boldsymbol{a}_\Delta}, \widetilde{r_\Delta}), (\boldsymbol{s}'_1, \boldsymbol{a}'_1, r'_1), \ldots, (\boldsymbol{s}'_{T'}, \boldsymbol{a}'_{T'}, r'_{T'})\}$ to enhance the offline dataset. We put the augmented trajectories into the augmented dataset $\mathcal{D}_{\text{aug}}$.

Mixing generated data with sampled data

In training offline RL algorithms, we sample data from $D^* = D \cup D_{\text{aug}}$ with a fixed ratio

r= (number of original data : number of augmented data)

between the original data from D and the augmented data from D_{aug} in each training batch.

Evaluation

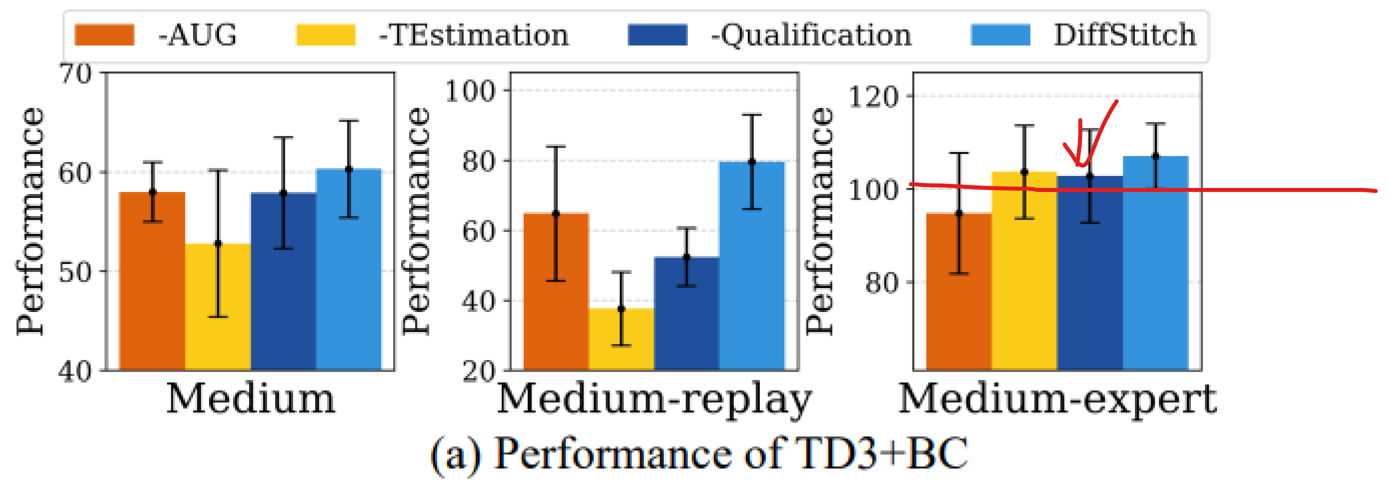
Methods:

1. **TD3+BC is a imitation learning method, which jointly learns the policy and value function**
2. **DT(Decision transformer**) DT is a trajectory optimization method, which takes trajectory data as input and applies Transformer (Vaswani et al., 2017) to model the distribution of trajectories in the offline dataset.

Data:

1. For Decision Transformer(DT), we train it for 5 × 104 gradient steps on MuJoCo-Gym tasks and 105 gradient steps on Adroit-pen tasks.

Ablation



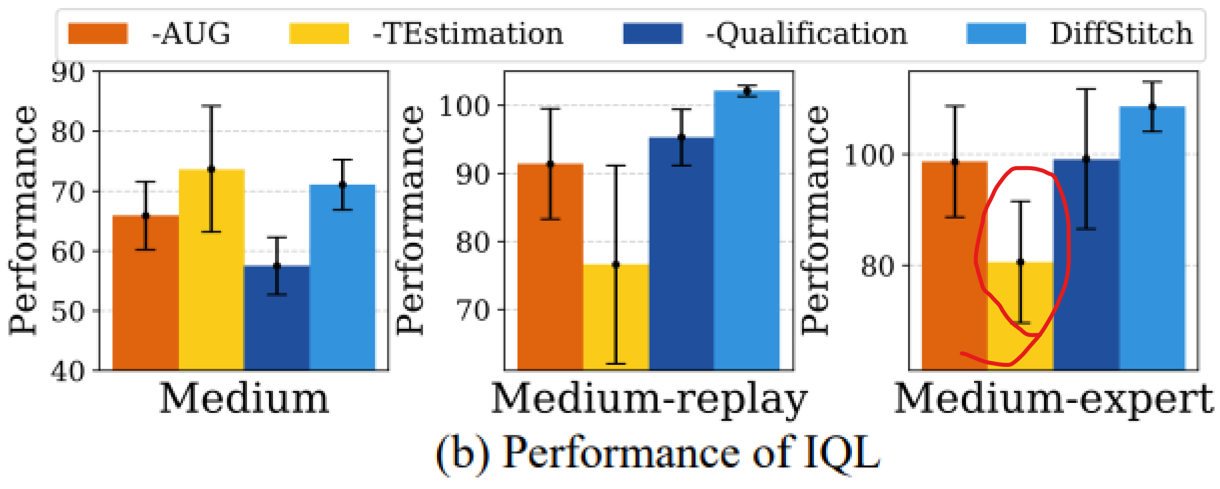


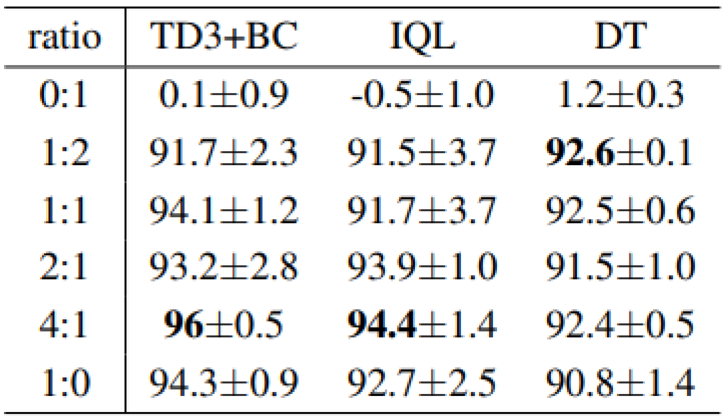
Figure 5. Ablation study on the hopper

Hyperparameters

Data ratio r (original/generated) is affected by training method. The lower the optimal r is, the more scalable our data augmentation is.

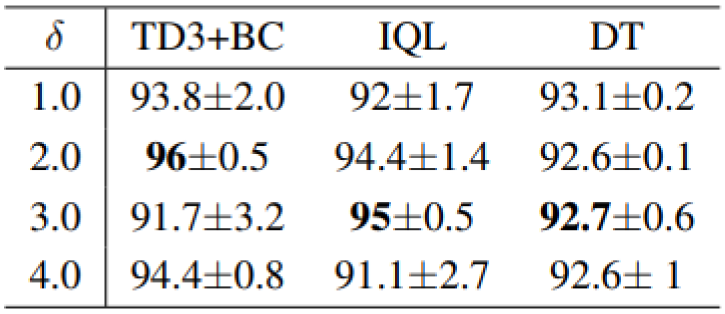
Qualification threshold δ determines whether we discard generated samples that deviates from the dynamics model's prediction==discrepancy with the original data. The higher δ is, the more robust our generation scheme.

Table 2. The impact of data ratio r.



ratio	TD3+BC	IQL	DT
0:1	0.1±0.9	-0.5±1.0	1.2±0.3
1:2	91.7±2.3	91.5±3.7	**92.6**±0.1
1:1	94.1±1.2	91.7±3.7	92.5±0.6
2:1	93.2±2.8	93.9±1.0	91.5±1.0
4:1	**96**±0.5	**94.4**±1.4	92.4±0.5
1:0	94.3±0.9	92.7±2.5	90.8±1.4

Table 3. The impact of qualification threshold δ.



δ	TD3+BC	IQL	DT
1.0	93.8±2.0	92±1.7	93.1±0.2
2.0	**96**±0.5	94.4±1.4	92.6±0.1
3.0	91.7±3.2	**95**±0.5	**92.7**±0.6
4.0	94.4±0.8	91.1±2.7	92.6± 1

In this sense, we gonna use Decision Transformer.