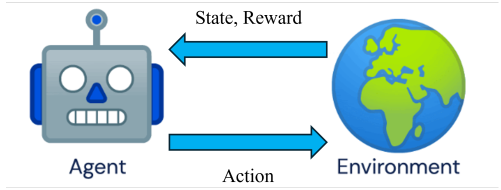
Tackling Data Corruption in Offline Reinforcement Learning via Sequence Modeling

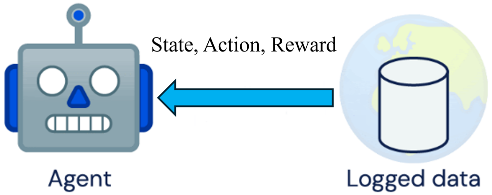
Jiawei Xu[1*] Rui Yang[2*] Shuang Qiu[3] Feng Luo[4]
Meng Fang[5] Baoxiang Wang[1] Lei Han[6]

[1]The Chinese University of Hong Kong, Shenzhen [2]University of Illinois Urbana-Champaign
[3]City University of Hong Kong [4]Rice University [5]University of Liverpool [6]Tencent Robotics X

March 22, 2025



(a) Online Reinforcement Learning



(b) Offline Reinforcement Learning

Figure: Comparison of Online vs. Offline Reinforcement Learning.

Motivating Example

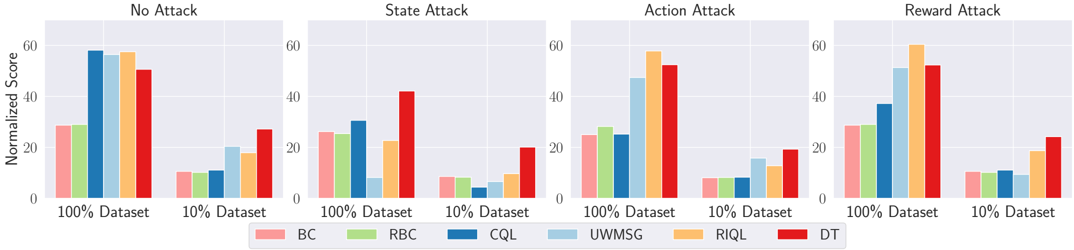


Figure: Average normalized scores of offline RL algorithms under random data corruption across three MuJoCo tasks (halfcheetah, walker2d, and hopper). Many offline RL algorithms experience substantial performance declines when subjected to data corruption. **In contrast, DT demonstrates remarkable robustness, particularly in the 10% data regime.**

Key Question

Decision Transformer (DT) [1] exhibits remarkable robustness to data corruption, especially in scenarios involving limited dataset conditions, even without any additional robustness techniques.

How can we further unleash the potential of sequence modeling in addressing data corruption with limited datasets in offline RL?

Robust Decision Transformer (RDT)

We propose **R**obust **D**ecision **T**ransformer (**RDT**) by incorporating three simple yet effective robust techniques:

- **Embedding dropout** to improve the model's robustness against erroneous inputs.
- **Gaussian weighted learning** to mitigate the effects of corrupted labels.
- **Iterative data correction** to eliminate corrupted data from the source.

Embedding Dropout

To against erroneous inputs:

- Randomly drop raw elements.
  $\Rightarrow$ Loss of much information. ✖
- Randomly drop dimensions of raw elements.
  $\Rightarrow$ Loss of much information for elements (such as actions) with low dimensions. ✖
- Randomly drop dimensions from feature space.
  $\Rightarrow$ Learn more robust embedding representations while preventing overfitting. ✔

Therefore, we apply **Embedding Dropout** techniques to learn robust embeddings.

Embedding Dropout - Supportive Experiment

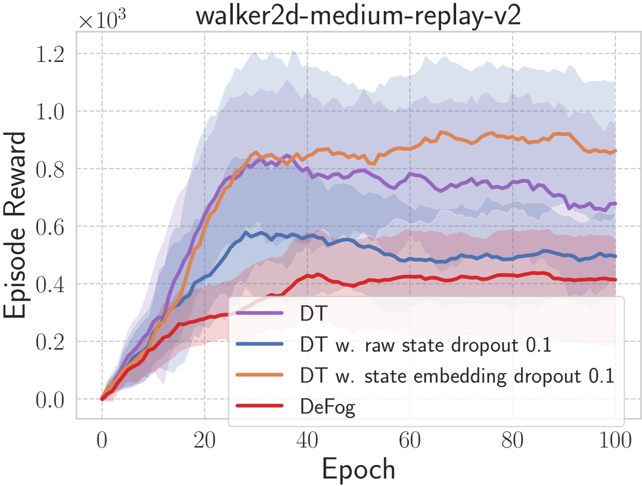


Figure: Comparison results under state attack. Embedding dropout outperforms directly dropping the entire state (DeFog [2]) or dropping dimensions on the raw state.

Gaussian weighted learning

To softly reduce the effect of corrupted labels, we use the **Gaussian weight**, i.e., a weight that decays exponentially in accordance with the sample's loss. This is mathematically formulated as:

$$w_{a_t} = e^{-\beta \cdot \delta_{a_t}^2}, \text{ where } \delta_{a_t} = \text{no\_grad}(\|\pi_\theta(\hat{\tau}_{t-K+1:t-1}, \hat{R}_t, \hat{s}_t) - \hat{a}_t\|_2).$$

The δ_{a_t} represents prediction errors with detached gradients. The variable $\beta \geq 0$ acts as the temperature coefficient, providing flexibility to control the "blurring" effect of the Gaussian weights.

Gaussian weighted learning

The loss function of RDT is expressed as follows:

$$\mathcal{L}_{RDT}(\theta) = \mathbb{E}_{\hat{\tau}\sim\mathcal{D}}\left[\frac{1}{K}\sum_{t=0}^{K-1} w_{a_t}(\pi_\theta(\hat{\tau}_{t-K+1:t-1}, \hat{R}_t, \hat{s}_t) - \hat{a}_t)^2\right].$$

Gaussian weighted learning enables us to mitigate the detrimental effects of corrupted labels, thereby enhancing the algorithm's robustness.

Gaussian Weighted Learning - Supportive Experiment

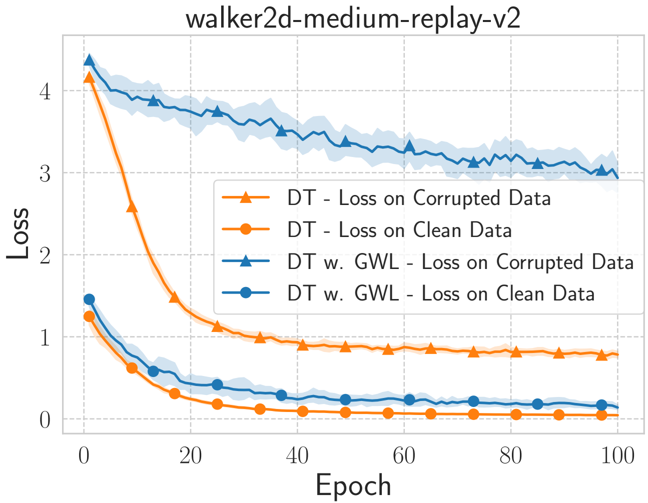


Figure: We observe that corrupted labels typically lead to a larger loss. Gaussian weighted learning (DT w. GWL) alleviates overfitting to the corrupted data.

Iterative Data Correction

We propose iteratively correcting corrupted data in the dataset using the model's predictions to bring the data closer to their true values in the next iteration. Take the actions as an example:

1. Store the distribution information of prediction error δ from the loss function throughout the learning phase by preserving the mean μ_δ and variance σ_δ^2.
2. To detect corrupted actions, calculate the z-score, denoted by $z = \frac{\delta - \mu_\delta}{\sigma_\delta}$, for each sampled action $\hat{a}$.
3. If the condition $z > \zeta \cdot \sigma_\delta$ (ζ is the predefined detection threshold) is met for any given action $\hat{a}$, then we infer that the action $\hat{a}$ has been corrupted and permanently replace $\hat{a}$ in the dataset with the predicted action.

Iterative data correction - Supportive Experiment

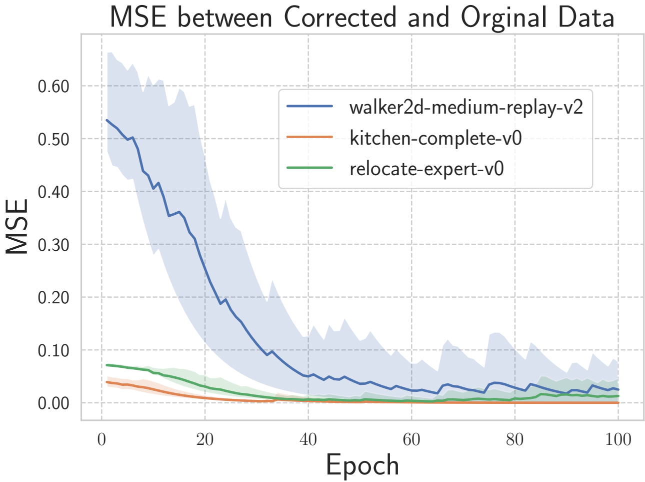


Figure: The MSE between corrected and original clean data gradually decreases to near zero.

Overview

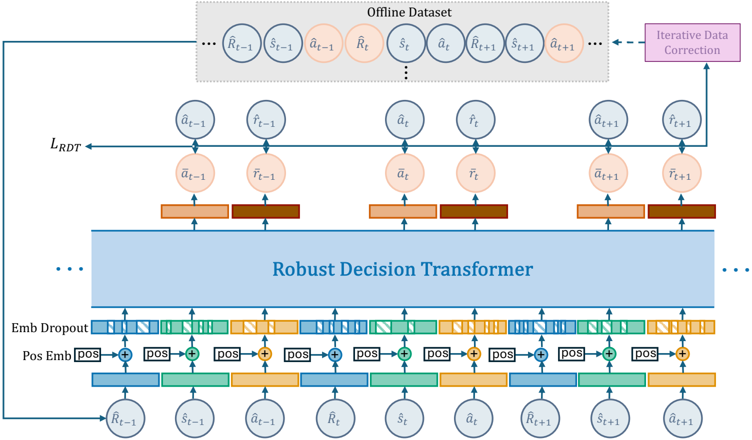


Figure: RDT enhances the robustness of DT against data corruption by incorporating three components: embedding dropout, Gaussian weighted learning, and iterative data correction.

Settings

- **Benchmark [3]:** MuJoCo, Kitchen, and Adroit.
- **Limited Dataset:** We randomly select 10% and 1% of the trajectories from MuJoCo and Adroit tasks respectively, labeled as MuJoCo (10%) and Adroit (1%). The Kitchen dataset is not downsampled due to its already limited size.

Data Corruption during Training

We follow previous work [4] to design the data corruption:

- **Random Corruption**
  It adds random noise to the affected elements in the datasets. For example, the corrupted state $\hat{s}_t = s_t + \lambda \cdot \mathrm{std}(s)$, where $\lambda \sim \mathrm{Uniform}[-\epsilon, \epsilon]^{\mathrm{d_s}}$ (d_s is the dimensions of state, and ϵ is the corruption scale) and $\mathrm{std}(s)$ is the d_s-dimensional standard deviation of all states in the offline dataset.
- **Adversarial Corruption**
  It uses Projected Gradient Descent attack [5] with pretrained value functions. Specifically, we introduce learnable noise to the states or actions and then optimize this noise by minimizing the pretrained value functions through gradient descent.

The corruption rate is set to 30%, and the corruption scale is $\epsilon = 1.0$.

Observation Perturbations during Testing

We evaluate RDT under two types of observation perturbations following prior works [6].

- **Random**: We create the perturbation set $\mathbb{B}_d(s, \epsilon) = \{\hat{s} \mid d(s, \hat{s}) \leq \epsilon\}$ for state s, where $d(\cdot)$ represents the l_∞ norm, and then sample one perturbed state.
- **Action Diff**: We sample 50 perturbed states within an l_∞ ball of norm ϵ and then select the one that maximizes the difference in actions: $\max_{\hat{s} \in \mathbb{B}_d(s,\epsilon)} ||\mu(s) - \mu(\hat{s})||^2$. Here, $\mu(s)$ denotes the pretrained deterministic policy on clean dataset.

Comparison Results under Random Corruption

Table: Results under random data corruption.

Attack	Task	BC	DeFog	RIQL	DT	**RDT**
State	MuJoCo (10%)	8.6	12.9	9.7	20.1	**23.1**
	KitChen	16.8	19.2	28.3	33.3	**43.8**
	Adroit (1%)	57.8	69.7	38.1	84.7	**93.9**
	Average	27.7	33.9	25.4	46.0	**53.6**
Action	MuJoCo (10%)	8.1	19.0	12.8	19.3	**29.5**
	KitChen	22.8	15.3	31.5	23.1	**43.1**
	Adroit (1%)	49.1	81.8	39.6	73.2	**96.5**
	Average	26.7	38.7	28.0	38.5	**56.3**
Reward	MuJoCo (10%)	10.6	17.1	18.6	24.2	**31.5**
	KitChen	36.3	22.0	46.3	44.6	**54.9**
	Adroit (1%)	69.7	85.7	39.1	83.8	**104.1**
	Average	38.9	41.6	34.7	50.9	**63.5**
Average over all tasks		31.1	38.1	29.3	45.1	**57.8**

Comparison Results under Adversarial Corruption

Table: Results under adversarial data corruption.

Attack	Task	BC	DeFog	RIQL	DT	**RDT**
State	MuJoCo (10%)	9.9	12.9	11.0	21.9	**23.6**
	KitChen	23.4	20.0	40.4	37.9	**49.1**
	Adroit (1%)	60.2	75.6	44.5	85.3	**95.4**
	Average	31.2	36.2	32.0	48.4	**56.0**
Action	MuJoCo (10%)	4.2	10.3	7.1	13.4	**21.6**
	KitChen	6.2	3.9	8.0	5.4	**34.0**
	Adroit (1%)	8.4	51.8	42.4	47.4	**80.3**
	Average	6.3	22.0	19.2	22.1	**45.3**
Reward	MuJoCo (10%)	10.6	11.5	18.3	25.2	**31.9**
	KitChen	36.3	20.3	45.9	48.1	**56.0**
	Adroit (1%)	69.7	80.8	53.6	90.9	**96.5**
	Average	38.9	37.6	39.2	54.7	**61.5**
Average over all tasks		25.5	31.9	30.1	41.7	**54.3**

Comparison Results under Mixed Corruption

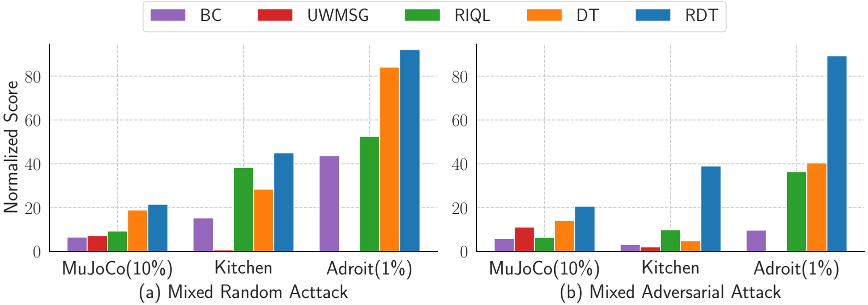
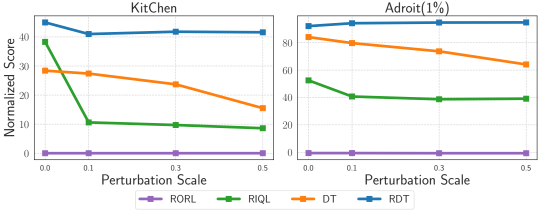


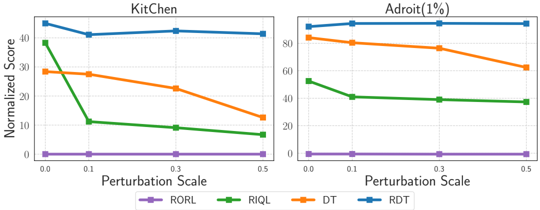
Figure: Results under (a) mixed random corruption and (b) mixed adversarial corruption.

We conduct experiments under mixed data corruption settings, where all three elements (states, actions, and rewards) are corrupted.

Comparison Results under Observation Perturbations during Testing



(a) Random observation perturbation.



(b) Action Diff observation perturbation.

Figure: Results under various observation perturbation scales.

Conclusion

Conclusion:

- **Robustness:** RDT excels against various data corruption types.
- **Superiority:** RDT performs well in handling training and testing attacks.
- **Inspiration:** Encourages sequence modeling for complex corruption scenarios.

References I

[1] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[2] Kaizhe Hu, Ray Chen Zheng, Yang Gao, and Huazhe Xu. Decision transformer under random frame dropping. *arXiv preprint arXiv:2303.03391*, 2023.

[3] 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.

[4] Rui Yang, Han Zhong, Jiawei Xu, Amy Zhang, Chongjie Zhang, Lei Han, and Tong Zhang. Towards robust offline reinforcement learning under diverse data corruption. In *The Twelfth International Conference on Learning Representations*, 2024.

References II

[5] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[6] Rui Yang, Chenjia Bai, Xiaoteng Ma, Zhaoran Wang, Chongjie Zhang, and Lei Han. Rorl: Robust offline reinforcement learning via conservative smoothing. *Advances in Neural Information Processing Systems*, 35:23851–23866, 2022.

Thanks!