Rein*for*mer: Max-Return Sequence Modeling for Offline RL

Zifeng Zhuang [1 2] **Dengyun Peng** [2 3] **Jinxin Liu** [2] **Ziqi Zhang** [2] **Donglin Wang** [2]

Abstract

As a data-driven paradigm, offline reinforcement learning (RL) has been formulated as sequence modeling that conditions on the hindsight information including returns, goal or future trajectory. Although promising, this supervised paradigm overlooks the core objective of RL that maximizes the return. This overlook directly leads to the lack of trajectory stitching capability that affects the sequence model learning from sub-optimal data. In this work, we introduce the concept of max-return sequence modeling which integrates the goal of maximizing returns into existing sequence models. We propose **Rein*for*ced Trans*for*mer** (**Rein*for*mer**), indicating the sequence model is reinforced by the RL objective. **Rein*for*mer** additionally incorporates the objective of maximizing returns in the training phase, aiming to predict the maximum future return within the distribution. During inference, this in-distribution maximum return will guide the selection of optimal actions. Empirically, **Rein*for*mer** is competitive with classical RL methods on the D4RL benchmark and outperforms state-of-the-art sequence model particularly in trajectory stitching ability. Code is public at `https://github.com/Dragon-Zhuang/Reinformer`.

1. Introduction

In classical online reinforcement learning (RL), the agent interacts with the environment to collect data and then uses that to derive the policy which maximizes the returns (Sutton et al., 1998). Mainstream RL algorithms rely on fitting optimal value functions (Watkins & Dayan, 1992) or calculating policy gradients (Sutton et al., 1999). Both in terms of paradigms and algorithms, reinforcement learning differs significantly from data-driven supervised learning. To avoid expensive or even risky online interaction and reuse pre-collected datasets, offline RL is proposed. Compared to classical RL with environment interaction, offline RL shares a paradigm more akin to supervised learning due to learning from datasets. This encourages researchers to explore offline algorithms from the supervised perspective.

Decision Transformer (DT) (Chen et al., 2021) maximizes the likelihood of actions conditioned on the historical trajectories (including returns), which essentially converts offline RL to a supervised sequence modeling. Lots of subsequent works have improved DT from different perspectives, including model architecture (Kim et al., 2023), online finetuning (Zheng et al., 2022), unsupervised pretraining (Xie et al., 2023) and stitching ability (Wu et al., 2023). However, these supervised paradigms seem to overlook the fundamental objective of reinforcement learning that is to maximize returns. The only naive approach to maximize return is to manually provide an initial return that is as large as possible. This approach is acceptable in some cases, but it becomes fatal when emphasizing trajectory stitching (Brandfonbrener et al., 2022). A typical example, also Figure 2, is the stitching between a fail trajectory ($R = 0$: *starting from the initial point but not reaching the goal*) and a successful trajectory ($R = 1$: *reaching the goal but not starting from the initial point*). Ideal returns should be 0 at first and then switch to 1 when stitching to the successful trajectory, which is conflict with the naive max approach that manually sets 1.

In this work, we propose the concept of max-return sequence modeling, a supervised paradigm that integrates the RL objective. Max-return sequence modeling not only maximizes the likelihood of actions, but also predicts the maximum in-distribution returns. Concretely, expectile regression (Sobotka & Kneib, 2012; Aigner et al., 1976) is adopted to make the predicted returns as close as possible to the maximum returns that are achievable under the current historical trajectory. When performing inference, the sequence model first predicts the current maximum return and then selects the best action from the offline dataset distribution, guided by this predicted maximized return. An implementation of max-return sequence modeling is **Rein*for*ced Trans*for*mer** (**Rein*for*mer**), representing the sequence model reinforced by the maximum return objective. When facing trajectory stitching in Figure 2, **Rein*for*mer** tends to predict 0 at the initial point and predict 1 when switching to the successful

[1]Zhejiang University [2]School of Engineering, Westlake University [3]Harbin Institute of Technology. Correspondence to: Donglin Wang <wangdonglin@westlake.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

trajectory due to in-distribution max return prediction.

We exhaustively evaluate **Rein*former*** on `Gym`, `Maze2d`, `Kitchen` and `Antmaze` datasets from the D4RL benchmark (Fu et al., 2020). **Rein*former*** has achieved performance that is competitive with classical offline RL algorithms. Compared with state-of-the-art sequence models, **Rein*former*** exhibits promising improvement, especially on datasets where trajectory stitching ability is highly demanded to learn from sub-optimal data. Our further analysis and ablation study elucidates the role of the return loss and the characteristics of the predicted maximized returns.

2. Preliminaries

2.1. Offline Reinforcement Learning

Reinforcement Learning (RL) is a framework of sequential decision. Typically, this problem is formulated by a Markov decision process (MDP) $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, r, p, d_0, \gamma\}$, with state space $\mathcal{S}$, action space $\mathcal{A}$, scalar reward function r, transition dynamics p, initial state distribution $d_0(s_0)$ and discount factor γ (Sutton et al., 1998). The objective of RL is to learn a policy, which defines a distribution over action conditioned on states $\pi(a_t|s_t)$ at timestep t, where $a_t \in \mathcal{A}$, $s_t \in \mathcal{S}$. Given this definition, the trajectory $\tau = (s_0, a_0, \cdots, s_T, a_T)$ generated by the agent's interaction with environment $\mathcal{M}$ can be described as a distribution $P_\pi(\tau) = d_0(s_0) \prod_{t=0}^{T} \pi(a_t|s_t)\, p(s_{t+1}|s_t, a_t)$, where T is the length of the trajectory, and it can be infinite. The goal of RL is to find a policy π that maximizes the expectation of the discounted cumulative return under the trajectory distribution $J(\pi) = \mathbb{E}_{\tau \sim P_\pi(\tau)} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right]$.

For **offline** reinforcement learning (Levine et al., 2020), the interaction with the environment $\mathcal{M}$ is forbidden and only a fixed offline dataset full of transitions is provided $\mathcal{D} = \left\{ (s_t, a_t, r_t, s_{t+1}, a_{t+1})_{t=1}^{N} \right\}$ where $r_t \doteq r(s_t, a_t)$. This setting is more challenging since the agent is unable to explore the environment and collect additional feedback.

2.2. Sequence Modeling in Reinforcement Learning

Compared to online reinforcement learning which interacts with the environment, offline RL is more similar to the data-driven paradigm given the offline dataset $\mathcal{D}$. As a result, offline RL (Chen et al., 2021) has been formulated as the supervised sequence modeling, different from the classical MDP formulation. The offline dataset can be denoted as the sequence form rather than transitions $\mathcal{D} = \left\{ \left(\cdots, s_t^{(n)}, a_t^{(n)}, g_t^{(n)} \cdots \right) \right\}$. Here $g_t^{(n)}$ is the returns-to-go (or simply returns) defined as $g_t^{(n)} \doteq \sum_{t'=t}^{T} r\left(s_{t'}^{(n)}, a_{t'}^{(n)}\right)$ that represents the sum of future rewards from current timestep t. Decision Transformer (DT) (Chen et al., 2021), following the upside-down RL (Srivastava et al., 2019; Schmidhuber, 2019), predicts the actions based on the previous trajectories τ concatenated with returns-to-go $g_t^{(n)}$:

$$\mathcal{L}_{\mathrm{DT}} = \mathbb{E}_{t,n}\left[a_t^{(n)} - \pi_{\mathrm{DT}}\left(\langle g, s, a \rangle_{t-K}^{(n)}; g_t^{(n)}, s_t^{(n)} \right) \right]^2,$$

where $\mathbb{E}_{t,n}$ is an omission of $\mathbb{E}_{t\in[0,T],n\in[1,N]}$. Besides, $\langle g, s, a \rangle_{t-K}^{(n)}$ denotes the previous K timesteps trajectory supplemented with returns-to-go $g_t^{(n)}$ and $\langle g, s, a \rangle_{t-K}^{(n)} = \left(g_{t-K+1}^{(n)}, s_{t-K+1}^{(n)}, a_{t-K+1}^{(n)}, \cdots, g_{t-1}^{(n)}, s_{t-1}^{(n)}, a_{t-1}^{(n)} \right)$. In DT, the policy π_{DT} is implemented by a causal transformer, namely the decoder layers. For each timestep t, three different tokens containing returns-to-go, state and action $g_t^{(n)}, s_t^{(n)}, a_t^{(n)}$ are fed into the model. And the future action $\hat{a}_t^{(n)}$ is predicted via autoregressive modeling.

To enable online finetuning ability, Online Decision Transformer (ODT) (Zheng et al., 2022) stochastic models the action as a Gaussian distribution and trains the model by maximizing action likelihood and another max-entropy term:

$$\begin{aligned}\mathcal{L}_{\mathrm{ODT}} =& \mathbb{E}_{t,n}\Big[-\log \pi_{\mathrm{ODT}}\left(a_t^{(n)} | \langle g, s, a \rangle_{t-K}^{(n)}; g_t^{(n)}, s_t^{(n)} \right) \\ & - \lambda H\left(\pi_{\mathrm{ODT}}\left(\cdot | \langle g, s, a \rangle_{t-K}^{(n)}; g_t^{(n)}, s_t^{(n)} \right) \right) \Big],\end{aligned} \tag{1}$$

where λ is the temperature parameter (Haarnoja et al., 2018) and λ will be adaptively updated by another temperature loss $\mathcal{L}_\lambda = \lambda\left(H\left(\pi_{\mathrm{ODT}}\left(\cdot | \langle g, s, a \rangle_{t-K}^{(n)}; g_t^{(n)}, s_t^{(n)} \right) \right) - \beta \right)$ with β is the prefixed value [1].

For the **Inference** of DT and ODT, one desired performance $\hat{g}_0$ must be specified as returns-to-go at first. Along with the initial environment state s_0, the next action will be generated by the model $a_1 = \pi_{\mathrm{DT}}(\hat{g}_0, s_0)$ or $\pi_{\mathrm{ODT}}(a_1|\hat{g}_0, s_0)$. Once the action a_1 is executed by the environment, the next state s_1 and reward r_1 are returned. Then the next returns-to-go should minus the returned reward $\hat{g}_1 = \hat{g}_0 - r_1$. This process is repeated until the episode terminates.

Drawbacks: These sequence models mainly focus on maximizing the action likelihood while neglecting the RL objective that maximizes the returns. Manually setting a large initial $\hat{g}_0$ can be seen as a naive approach to maximize returns. Some cases are reasonable, but in the context of trajectory stitching, this method will lead to severe out-of-distribution (OOD) issues. It is crucial to consider the goal of maximizing returns within the framework of sequence modeling and to derive the maximum in-distribution returns during the inference phase.

[1] Usually, this parameter is the negative value of the action dimension $\beta = -\dim(\mathcal{A})$.

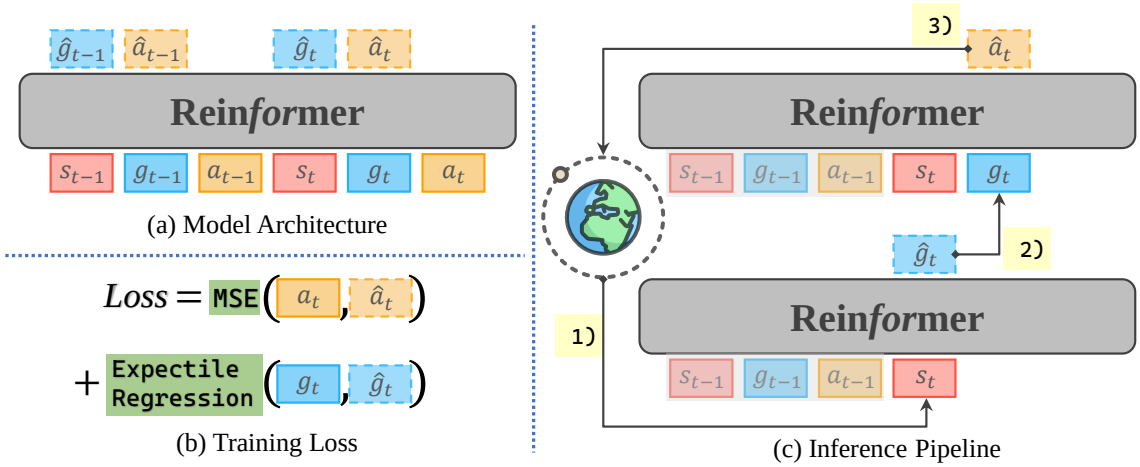


Figure 1. The overview of **ReinForced** Trans**Former** (**Rein*former***) (a) Model Architecture: The returns-to-go is the second token of **Rein*former*** inputs and the outputs contain returns and actions. (b) Train Loss: As a max-return sequence model, **Rein*former*** not only maximizes the action likelihood but also maximizes returns by expectile regression. (c) Inference Pipeline: When inference, **Rein*former*** first 1) gets state from the environment to predict the in-distribution maximum return. Then 2) predicted in-distribution max return is concatenated with state to predict the optimal action. Finally, 3) the environment executes the predicted action to return the next state.

3. Rein*former*: Reinforced Transformer

In this section, we start with a simple maze example to illustrate why classical sequence models and the naive max-return approach are unlikely to solve the trajectory stitching problem. Further, we introduce the concept of max-return sequence modeling and theoretically demonstrate that this paradigm can derive maximum returns without suffering from OOD issues. Finally, we present the implementation details of our sequence model **ReinForced** Trans**Former** (**Rein*former***) from three aspects: model architecture, the loss function during training and the inference pipeline.

3.1. Trajectory Stitching Example

In the offline RL literature, trajectory stitching receives lots of attention. Ideally, the offline agent should take suboptimal trajectories that overlap and stitch them into the optimal trajectories (Kostrikov et al., 2021b; Liu et al., 2023). It has been proved both theoretically (Brandfonbrener et al., 2022) and empirically (Kumar et al., 2022) that the return-conditioned sequence modeling lacks stitching ability. We utilize the following example to thoroughly describe this.

Example The Figure 2 depicts a toy maze, where s_0 is the starting point, s_G is the final goal state with reward $r = 1$, s_1 is a boom with $r = -1$ and other states are all $r = 0$. The offline dataset contains two trajectories one trajectory τ_1 starts from the initial point s_0 but doesn't reach the goal while another τ_2 reaches the goal s_G but doesn't start from s_0. Trajectory stitching expects the offline agent can follow the first half of τ_1 (from starting point s_0 to s_4) and then take the second half of τ_2 (from s_4 to the goal s_G) to reach the goal. We first explain why the basic sequence model, such as DT, might fail.

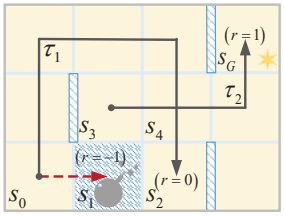


Figure 2. A maze example for trajectory stitching analysis.

For DT, if we set initial returns-to-go as $\hat{g}_0 = 0$ at the starting point, the offline agent will smoothly reach the intersection state s_4. However, since returns-to-go is still zero $\hat{g}_4 = 0$, DT will reach the state s_2 rather then s_G. Only when $\hat{g}_4 = 1$, DT is possible to follow τ_2. But $\hat{g}_4 = 1$ is *impossible* to achieve given $\hat{g}_0 = 0$. If we apply the naive max approach and set the initial $\hat{g}_0 = 1$, the agent might directly walk towards the boom s_1 $(r = -1)$ because $\hat{g}_0 = 1$ is the OOD returns-to-go for the starting point[2].

If the sequence model is endowed with capability to maximize the returns like RL, Let's see what might happen. At the starting point s_0, only τ_1 is contained in dataset so the model will predict $\hat{g}_0 = 0$. When offline agent comes to the intersection s_4, the latter segments of both trajectories are available. If the sequence model is able to maximize return, then τ_2 is more likely to be selected since the return $R = 1$ is larger. This inspires us to bring the capability of maximizing returns back into sequence modeling.

3.2. Max-Return Sequence Modeling

The key objective of RL is to obtain the optimal action for the current state through maximizing the returns. We aim to equip supervised sequence modeling with additional maximizing return objective. And during inference, the sequence model can select optimal action conditioned on

[2]Actually, DT or ODT use this naive approach for Antmaze dataset, called the "delayed" mode in code implementation.

the in-distribution maximized returns. We introduce the expectile regression as returns-to-go loss to achieve this.

Expectile regression (Newey & Powell, 1987) is well studied in applied statistics and econometrics and has been introduced into offline RL recently (Kostrikov et al., 2021b; Wu et al., 2023). Specifically, the returns-to-go loss based on the expectile regression is as follows:

$$\mathcal{L}_{\mathrm{g}}^{m}=\mathbb{E}_{t,n}\left[|m-\mathbb{1}\left(\Delta g<0\right)|\,\Delta g^{2}\right], \tag{2}$$

here $\Delta g=g_t^{(n)}-\hat{g}_t^{(n)}$ and $\hat{g}_t^{(n)}=\pi_\theta\left(\langle s,g,a\rangle_{t-K}^{(n)};s_t^{(n)}\right)$. It is noteworthy that the relative positions of these tokens are different from DT, where returns-to-go $g_t^{(n)}$ is placed after the state token $s_t^{(n)}$ and before the action $a_t^{(n)}$. Here $m\in(0,1)$ is the hyperparameter of expectile regression. When $m=0.5$, expectile regression degenerates into standard regression, also MSE loss. But when $m>0.5$, this asymmetric loss will give more weights to the $g_t^{(n)}$ larger than $\hat{g}_t^{(n)}$, which aligns with the asymmetric curves in Figure 3. Besides, The red arrow shows the weight increases as the m becomes larger. In other words, the predicted returns-to-go $\hat{g}_t^{(n)}$ will approach larger $g_t^{(n)}$.

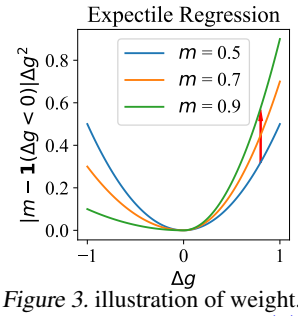


Figure 3. illustration of weight.

To unveil what the returns-to-go loss function has learned and offer a formal elucidation of its role, we introduce the following theorem along with its proof:

Theorem 3.1. *We first define* $\mathbf{S}_t^{(n)}\doteq\left[\langle s,g,a\rangle_{t-K}^{(n)};s_t^{(n)}\right]$. *For* $m\in(0,1)$, *denote* $\mathbf{g}^m\left(\mathbf{S}_t^{(n)}\right)=\pi_\theta^*\left(\mathbf{S}_t^{(n)}\right)$, *where* $\pi_\theta^*=\arg\min\mathcal{L}_g^m$, *then we have*

$$\lim_{m\to1}\mathbf{g}^m\left(\mathbf{S}_t^{(n)}\right)=g_{max},$$

where $g_{max}=\max_{\mathbf{a}\sim\mathcal{D}}g\left(\mathbf{S}_t^{(n)},\mathbf{a}\right)$ *denotes the maximum returns-to-go with actions from offline dataset.*

Proof. First, according to the monotonicity of expectile regression, we have $\mathbf{g}^{m_1}\le\mathbf{g}^{m_2}$ when $0<m_1<m_2<1$. Secondly, $\forall m\in(0,1)$, $\mathbf{g}^m\le g_{\max}$ and we can use contradiction to prove it. Assume one m_3 satisfies $\mathbf{g}^{m_3}>g_{\max}$, then all the returns-to-go from offline dataset will $g_t^{(n)}<\mathbf{g}^{m_3}$. The returns-to-go loss can be simplified given the same weight $1-m_3$:

$$\begin{aligned}\mathcal{L}_{\mathbf{g}}^{m_3}&=\mathbb{E}_{t,n}\left[(1-m_3)\left(g_t^{(n)}-\mathbf{g}^{m_3}\right)^2\right]\\&>\mathbb{E}_{t,n}\left[(1-m_3)\left(g_t^{(n)}-\max_{t,n}[g_t^{(n)}]\right)^2\right].\end{aligned}$$

This inequality holds because $g_t^{(n)}\le\max_{t,n}[g_t^{(n)}]<\mathbf{g}^{m_3}$. But this inequality is conflict with the fact that $\mathbf{g}^{m_3}$ is obtained by minimizing the returns-to-go loss. Therefore, the assumption is not valid and $\mathbf{g}^m\le g_{\max}$ is true.

Finally, this limit follows from the properties of bounded monotonically non-decreasing functions thus affirming the validity of the theorem. □

In one word, Theorem 3.1 indicates the loss $\mathcal{L}_{\mathrm{g}}^{m}$ will make the model predict the maximum returns-to-go when $m\to1$, which is similar to the maximizing returns objective in RL. When inference, the model will generate the action conditioned on this predicted maximum returns-to-go. Furthermore, the second step in our proof indicates that this predicted returns do not suffer from OOD issues.

3.3. Implementation Details

Now, we will focus on the specific implementation of **Rein*former***, describing the model input and output, training, and inference procedures. Figure 1 is an overview.

Model Architecture To accommodate the max-return sequence modeling paradigm, which predicts the maximum return and utilizes it as a condition to guide the generation of optimal actions, we have positioned returns between state and action. In detail, the input token sequence of **Rein*former*** and corresponding output tokens are summarized as follows:

$$\textbf{Input:}\quad\left\langle\cdots,s_t^{(n)},g_t^{(n)},a_t^{(n)}\right\rangle$$
$$\textbf{Output:}\quad\left\langle\hat{g}_t^{(n)},\hat{a}_t^{(n)},\square\right\rangle$$

here $g_t^{(n)}\doteq\sum_{t'=t}^{T}r\left(s_{t'}^{(n)},a_{t'}^{(n)}\right)$ is the returns-to-go and we will abbreviated it as *returns* in the absence of ambiguity. When predicting the $\hat{g}_t^{(n)}$, the model takes the current state $s_t^{(n)}$ and previous K timesteps tokens $\langle s,g,a\rangle_{t-K}^{(n)}$ into consideration. For the sake of simplicity, $\mathbf{S}_{t-K}^{(n)}$ denotes the input $\left[\langle s,g,a\rangle_{t-K}^{(n)};s_t^{(n)}\right]$. While the action prediction $\hat{a}_t$ is based on $\left(\mathbf{S}_{t-K}^{(n)},\mathbf{G}_{t-K}^{(n)}\right)=\left[\langle s,g,a\rangle_{t-K}^{(n)};s_t^{(n)},g_t^{(n)}\right]$. The □ represents this predicted token neither participates in training nor inference so we ignore it.

At the timestep t, different tokens are embedded by different linear layers and fed into the transformers (Vaswani et al.,

Algorithm 1 Training

Input: offline dataset $\mathcal{D}$, sequence model π_θ
for sample $\langle \cdots, s_t, g_t, a_t \rangle$ from $\mathcal{D}$ **do**
  $\hat{g}_t, \hat{a}_t = \pi_\theta(\cdots, s_t, g_t, a_t)$
  Calculate action loss $\mathcal{L}_\text{a}$ by Equation (3)
  Calculate returns-to-go loss $\mathcal{L}_\text{g}^m$ by Equation (4)
  Take gradient descent step on $\nabla_\theta\left(\mathcal{L}_\text{a} + \mathcal{L}_\text{g}^m\right)$
end for

Algorithm 2 Inference

Input: sequence model π_θ, environment Env
$s_0 = \text{Env}.reset(\,)$ and $t = 0$
repeat
  Predict maximum returns $\hat{g}_t = \pi_\theta(\cdots, s_t, \square, \square)$
  Predict optimal action $\hat{a}_t = \pi_\theta(\cdots, s_t, \hat{g}_t, \square)$
  $s_{t+1}, r_t = \text{Env}.step(\hat{a}_t)$ and $t = t + 1$
until done

2017) together. The output returns-to-go $\hat{g}_t^{(n)}$ is processed by a linear layer. For action $\hat{a}_t^{(n)}$, we adopt the Gaussian distribution and use the mean of this distribution for inference.

Training Loss Since the model predicts two parts, $\hat{g}_t$ and $\hat{a}_t$, the loss function is composed of returns-to-go loss and action loss. For the action loss, we adopt the loss function of ODT and simultaneously adjust the order of tokens:

$$\mathcal{L}_\text{a} = \mathbb{E}_{t,n}\Big[-\log\pi_\theta\left(a_t^{(n)}|\mathbf{S}_{t-K}^{(n)}, \mathbf{G}_{t-K}^{(n)}\right) - \lambda H\left(\pi_\theta\left(\cdot|\mathbf{S}_{t-K}^{(n)}, \mathbf{G}_{t-K}^{(n)}\right)\right)\Big] \tag{3}$$

The returns-to-go loss is the expectile regression with the parameter m:

$$\mathcal{L}_\text{g}^m = \mathbb{E}_{t,n}\left[|m - \mathbb{1}\left(\Delta g < 0\right)|\,\Delta g^2\right], \quad \text{with } \Delta g = g_t^{(n)} - \pi_\theta\left(\mathbf{S}_{t-K}^{(n)}\right). \tag{4}$$

Two loss functions have the same weight so the total loss is $\mathcal{L}_\text{a} + \mathcal{L}_\text{g}$. Algorithm 1 summarizes the training process.

Inference Pipeline For each timestep t, the action is the last token, which means the predicted action is affected by state from the environment and the returns-to-go. The returns of the trajectories output by the sequence model exhibit a positive correlation with the initial conditioned returns-to-go (Chen et al., 2021; Zheng et al., 2022). That is, within a certain range, higher initial returns-to-go typically lead to better actions. In classical Q-learning (Mnih et al., 2015), the optimal value function Q^* can derive the optimal action a^* given the current state. In the context of sequence modeling, we also assume that the maximum returns-to-go are required to output the optimal actions. The inference pipeline of the **Rein*former*** is summarized as follows:

$$\overset{\text{Env}}{\longmapsto} s_0 \xrightarrow{\pi_\theta} g_0 \xrightarrow{\pi_\theta} a_0 \xrightarrow{\text{Env}} s_1 \xrightarrow{\pi_\theta} g_1 \xrightarrow{\pi_\theta} a_1 \to \cdots \tag{5}$$

Specially, the environment initializes the state s_0 and then the sequence model π_θ predicts the maximum returns-to-go g_0 given current state s_0. Concatenating g_0 with s_0, π_θ can output the optimal action a_0. Then the environment transitions to the next state s_1 and the reward r_1. It should be noted that this reward r_1 will **not** participate in the inference. Repeat the above steps until the trajectory comes to an end. This pipeline has been summarized in Algorithm 2.

3.4. Comparison with EDT

Elastic Decision Transformer (EDT) (Wu et al., 2023) explicitly considers trajectory stitching, so we briefly introduce EDT and then compare it with our **Rein*former***. EDT also introduces the same returns-to-go loss (4). During the inference phase, EDT uses this loss to estimate the maximum returns-to-go $\hat{g}_\text{max}(K)$ achievable under different historical lengths K. EDT then selects the maximum $\hat{g}_\text{max}^* = \max_K \hat{g}_\text{max}(K)$ and finally determine the action based on the expert action inference (Lee et al., 2022) along with the EDT model. The inference of EDT dynamically adjusts historical trajectories, which preserves longer historical lengths when previous trajectories are optimal and shortens them when they are suboptimal.

Similarly, EDT requires initial returns as input. Therefore, we also categorize EDT as a kind of basic sequence model that does not explicitly consider maximizing return. Although EDT is equipped with the same returns-to-go loss, it still relies on the naive max-return. For the `Antmaze` dataset that requires trajectory stitching, EDT does not provide experimental results. Our reproduced results indicate that EDT perform poorly on `Antmaze`.

4. Related Work

Offline Reinforcement Learning As the cradle of supervised sequence modeling, offline RL (Levine et al., 2020) breaks free from the traditional paradigm of online interaction. Classic off-policy actor-critic algorithms (Degris et al., 2012; Konda & Tsitsiklis, 1999) encounter out-of-distribution (OOD) issues, where the value function tends to overestimate the OOD state-action pairs (Fujimoto et al., 2019; Liu et al., 2024c). Mainstream offline algorithms can be categorized into two classes. One is policy constraint, which constrains the learned policy stay close to the behavior policy based on different "distance" such as batch constrained (Fujimoto et al., 2019), KL divergence (Wu et al., 2019; Liu et al., 2022), MMD distance (Kumar et al., 2019) and MSE constraint (Fujimoto & Gu, 2021). The

Table 1. The normalized last score on D4RL `Gym-v2`, `Maze2d-v1` and `Kitchen-v0` dataset. We report the mean and standard deviation of normalized score for five seeds. For each seed, the stats is calculated by 10 evaluation trajectories for `Gym` while 100 for `Maze2d` and `Kitchen`. The best result is **bold** and the blue result means the best result among sequence modeling.

	Dataset	Reinforcement Learning			Sequence Modeling				
		BC	CQL	IQL	DT	ODT	EDT	QDT	**Rein*former***
Gym	halfcheetah-medium	42.6	44.0	**47.4**	42.6	42.7	42.5	39.3	42.94±0.39
	halfcheetah-medium-replay	36.6	**45.5**	44.2	36.6	40.0	37.8	35.6	39.01±0.91
	halfcheetah-medium-expert	55.2	91.6	86.7	86.8		91.2		**92.04±0.32**
	hopper-medium	52.9	58.5	66.3	67.6	67.0	63.5	66.5	**81.60±3.32**
	hopper-medium-replay	18.1	**95.0**	94.7	82.7	86.6	89.0	52.1	83.31±3.47
	hopper-medium-expert	52.5	105.4	91.5	107.6		107.8		**107.82±2.14**
	walker2d-medium	75.3	72.5	78.3	74.0	72.2	72.8	67.1	**80.52±2.74**
	walker2d-medium-replay	26.0	**77.2**	73.9	66.6	68.9	74.8	58.2	72.89±5.06
	walker2d-medium-expert	107.5	108.8	**109.6**	108.1		107.9		109.35±0.32
	Total	*466.7*	*698.5*	*692.6*	*672.6*		*687.3*		***709.46***
Maze2d	maze2d-umaze	0.4	-8.9	42.1	18.1		35.8	**57.3**	57.15±4.27
	maze2d-medium	0.8	**86.1**	34.9	31.7		18.3	13.3	85.62±30.89
	maze2d-large	2.3	23.8	**61.7**	35.7		26.8	31.0	47.35±6.89
	Total	*3.5*	*101.0*	*138.7*	*85.5*		*85.9*	*101.6*	***190.12***
Kitchen	kitchen-complete	**65.0**	43.8	62.5			40.8		59.85±0.52
	kitchen-partial	38.0	49.8	46.3			10.0		**73.10±2.01**
	kitchen-mixed	51.5	51.0	51.0			9.8		**65.50±6.22**
	Total	*154.5*	*144.6*	*159.8*			*60.5*		***198.45***

other is value regularization, which regularizes the value function to assign low values on OOD state-action pairs (Liu et al., 2024b; Kostrikov et al., 2021a; Bai et al., 2022). Some algorithms take a different approach, understanding and solving offline RL problems from the perspective of on-policy learning. R-BVE (Gulcehre et al., 2021) and Onestep RL (Brandfonbrener et al., 2021) both transform off-policy style offline algorithms (such as CRR (Wang et al., 2020), BCQ (Fujimoto et al., 2019), BRAC (Wu et al., 2019)) into on-policy style. Besides, BPPO (Zhuang et al., 2023) finds the online algorithm PPO (Schulman et al., 2017) can directly solve the offline RL due to its inherent conservatism. In contrast, Decision Transformer (DT) (Chen et al., 2021) directly maximizes the action likelihood, which opens up a new paradigm called supervised sequence modeling.

Sequence Modeling in Reinforcement Learning Before Decision Transformer (DT), upside-down reinforcement learning (Srivastava et al., 2019; Schmidhuber, 2019) has already begun exploring RL solutions using supervised learning methods. DT (Chen et al., 2021) incorporates return as part of the sequence to predict the optimal action. This paradigm breaks away from the classic RL paradigm, circumventing OOD problems directly. Inspired by DT, RL is investigated from a supervised learning perspective, including network architecture (Kim et al., 2023; David et al., 2022), unsupervised pretraining (Xie et al., 2023) and large capacity model (Lee et al., 2022). While other works improve DT using RL components such as online finetuning (Zheng et al., 2022), trajectory stitching (Wu et al., 2023) and dynamics programming (Yamagata et al., 2023). However, no sequence model incorporates the RL objective that maximizes returns to enhance the model (Liu et al., 2024a).

5. Experiments

We conduct extensive experiments on `Gym`, `Maze2d`, `Kitchen` and `Antmaze` datasets from D4RL benchmark (Fu et al., 2020), aiming to answer the following questions:

- As a max-return sequence modeling method, how does **Rein*former*** compare with other state-of-the-art sequence models that do not explicitly consider maximizing returns? Can **Rein*former*** narrow the performance gap or even surpass classical offline RL algorithms?
- How does the predicted maximized returns-to-go and the parameter m of returns loss affect the **Rein*former*** performance? Additionally, what is the characteristic of the predicted maximized returns-to-go?

5.1. Results on D4RL Benchmark

We conduct a comprehensive evaluation of **Rein*former***, covering not only `Gym` dataset where performance approaches saturation but also more challenging datasets like `Maze2d`, `Kitchen` and `Antmaze`. Among these, `Antmaze` is characterized by sparse rewards and strongly emphasizes the stitching ability.

Baselines We compare our method with both representative reinforcement learning and state-or-the-art sequence modeling methods. Some results are reproduced using the official code and please refer to Appendix A for detail.

- **Reinforcement learning** includes Behavior Cloning (BC) (Pomerleau, 1988), Conservative Q-Learning (CQL) (Kumar et al., 2020) and Implicit Q-Learning (IQL) (Kostrikov et al., 2021b). Strictly, BC is an imitation learning algorithm. We categorize BC here just to emphasize that it is not a sequence modeling algorithm.
- **Sequence modeling** includes Decision Transformer (DT) (Chen et al., 2021), Online Decision Transformer (ODT) (Zheng et al., 2022), Elastic Decision Transformer (EDT) (Wu et al., 2023) and Q-learning Decision Transformer (QDT) (Yamagata et al., 2023).

Results on Dense Rewards Datasets The results of **Rein*former*** on three datasets are presented in Table 1. Evaluations of existing sequence modeling algorithms are notably insufficient, primarily focusing on `Gym` datasets while overlooking more challenging `Maze2d` and `Kitchen`. All the sequence model are able to achieve performance comparable to RL algorithms on `Gym` dataset. Furthermore, **Rein*former*** also demonstrates superior performance on `Maze2d` (+37.07%) and `Kitchen`(+28.45%) compared to the strongest baseline. Especially on `Kitchen`, classical offline algorithms do not significantly outperform BC, indicating that reward function is not fully utilized.

Results on Sparse Rewards Dataset The `Antmaze` dataset features sparse rewards, with $r = 1$ when reaching the goal. Both `medium-diverse` and `medium-play` does not contains complete trajectories from the starting point to the goal, which necessitates the algorithm to stitch failed trajectories to accomplish the goal. Despite the claims of trajectory stitching ability, our reproduced results in Table 2 indicate that EDT (Wu et al., 2023) performs poorly.

Table 2. The normalized best score on D4RL `Antmaze-v2` dataset. We report the mean and standard deviation of normalized score for five seeds. For each seed, the stats is calculated by 100 evaluation trajectories. The best result is **bold** and the blue result means the best result among sequence modeling.

Antmaze-v2	RL		Sequence Modeling			
	BC	IQL	DT	EDT	ODT	Rein*former*
umaze	68.5	84.0	64.5	67.8	53.1	**84.4±2.7**
umaze-diverse	64.8	**79.5**	60.5	58.3	50.2	65.8±4.1
medium-play	4.5	**78.5**	0.8	0.0	0.0	13.2±6.1
medium-diverse	4.8	**83.5**	0.5	0.0	0.0	10.6±6.9
Total	*142.6*	*325.5*	*126.3*	*126.1*	*103.3*	*174.0*

One observation in Table 2 is that the previous sequence models even underperform BC, which highlights the absence of trajectory stitching. **Rein*former*** exhibits a significant improvement compared to other sequence models, especially on `medium-diverse` and `medium-play`. However, compared to RL algorithm IQL (Kostrikov et al., 2021b), the performance gap remains quite considerable.

Summary and Discussion Thanks to the max-return sequence modeling paradigm, **Rein*former*** has emerged as the current state-of-the-art in sequence modeling. On `Antmaze` dataset that particularly demands extreme trajectory stitching, offline RL retains a significant advantage. However, on other datasets, our method demonstrates a noticeable improvement. Figure 4 describes the improvement probability (Agarwal et al., 2021) of **Rein*former***.

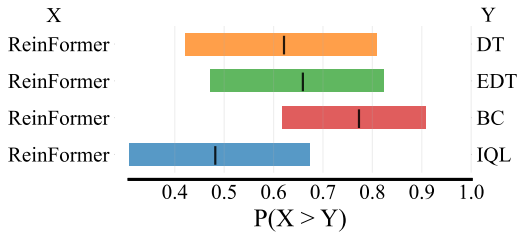


Figure 4. The probability of improvement of **Rein*former*** compared with other methods using rliable (Agarwal et al., 2021). The larger the probability is, the better our method performs.

5.2. Analysis of Returns-to-go loss

We will delve into the critical return loss in max-return sequence modeling. We aim to understand the impact of this module on the **Rein*former*** performance and explore the characteristics of the predicted maximized returns.

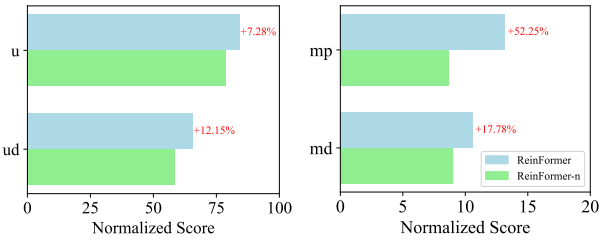
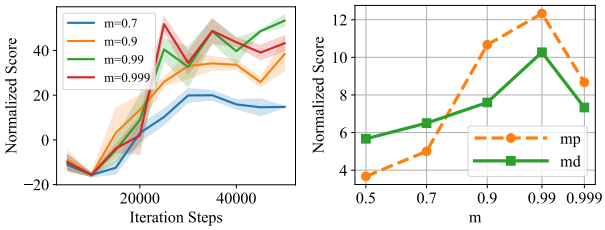


Figure 5. The comparison between different inference approaches on `Antmaze` dataset: **Rein*former*** with predicted maximized returns versus **Rein*former*-n** with naively maximized returns. Abbreviations: u → `umaze`, ud → `umaze-diverse`, mp → `medium-play`, md → `medium-diverse`.

Comparison on Maximizing Return Approach The key advantage of max-return sequence modeling lies in predicting the maximized in-distribution return during the inference. The compared baseline **Rein*former*-n** is based on the naive max approach that the initial returns is the max return in offline dataset. This baseline also includes the returns-to-go loss during training. As for the comparison results without return loss, please refer to the results in Table 1 or 2. We demonstrate the performance on different `Antmaze`

datasets in Figure 5. The improvement observed on `umaze` and `umaze-diverse`, which contains the entire trajectory from the starting point to the goal, is 19.43%. The improvement on `medium-play` and `medium-diverse`, where emphasizes trajectory stitching, is 70.03%.

Ablation Study on m Next, we analyze the impact of the hyper-parameter m in return loss. According to the theorem 3.1, when $m \to 1$, the learned returns will approach to the maximum return within the offline distribution. Furthermore, due to the fact that higher in-distribution return leads to better action, we can conclude that the performance will improve as m approaches 1. The experimental results in Figure 6 indeed align with the above theoretical analysis. Within a certain range, large m generally leads to better training process and higher performance. However, when $m = 0.999$ is excessively large, it may result in a performance decline. This could be attributed to the model overfitting to some extreme large returns in offline dataset.

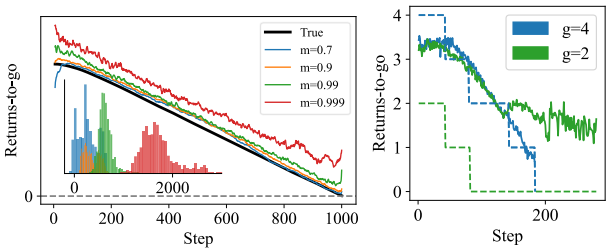


(a) `maze2d-umaze` (b) `Antmaze`

Figure 6. The performance given different hyper-parameter m. 5(a) illustrates the training curves with varying m on `maze2d-umaze` while 5(b) illustrates the trend of best results as m varies on `Antmaze-medium-play` (mp) and `medium-devise` (md).

Characteristic of the Predicted Maximized Returns We conduct experiments on `Halfcheetah-medium` and `Kitchen-complete` to analyze the characteristics of predicted maximized returns-to-go. In Figure 7(b), the dashed line represents the true returns-to-go. Original `Kitchen` environment is a staged sparse reward environment so the dashed line exhibits a step-wise decline. Trajectory return can only be one of $[0, 1, 2, 3, 4]$ but the dataset in D4RL (Fu et al., 2020) contains trajectories with returns ranging from 0 to 400. Therefore the predicted return, the solid line, is a continuous curve. The may results from some densification operation when constructing the dataset.

In environment with non-negative rewards, the true returns-to-go should exhibit a monotonically decreasing trend and eventually reach 0. In 7(a), the predicted returns with different m all exhibits clear decreasing trend and only the excessively large $m = 0.999$ is unable to converge to 0. For `Kitchen` in Figure 7(b), the solid curve $g = 4$ represents the predicted return of the trajectory that completes the task while $g = 2$ is the trajectory that completes the half. The curve $g = 4$ demonstrates a good monotonically decreasing property. But the descent rate of $g = 2$ significantly slowed down in the latter part and ultimately did not converge to zero. This high level return indicates that the model is still attempting to complete the task, although it ultimately fail.



(a) `Halfcheetah-medium` (b) `Kitchen-complete`

Figure 7. The predicted maximized returns-to-go by **Rein*former*** and the true returns-to-go obtained through interaction with the environment. In 6(a), the black curve is the mean of true returns-to-go of different m and the other curves are the predicted maximized returns-to-go. The bottom-left histogram illustrates the distribution of the differences between prediction and grand truth. In 6(b), the dashed line represents the true returns-to-go from the environment, while the solid line represents the predicted returns-to-go. It's worth noting that the trajectory return of `Kitchen` environment can only take one of $[0, 1, 2, 3, 4]$. Here, we present a trajectory that successfully completes the task ($g = 4$) along with another one that completes half of the task ($g = 2$).

From Figure 7(a), we observe that except for $m = 0.999$, the predicted returns can well match the actual returns. We plotted a frequency distribution histogram of the difference between the predicted return and the actual return. An observation is $m = 0.999$ suffers from overestimation although this does not harm the performance. In Figure 7(b), the case $g = 4$ exhibits a relatively high degree of matching. Because case $g = 2$ only completes the initial phase of the task, the predicted returns remains consistently high without any decrease or convergence. After translation, the true return still match the predicted maximized returns.

6. Conclusion, Discussion and Future work

In this work, we propose the paradigm of max-return sequence modeling which considers the RL objective that maximizes returns. Both theoretical analysis and experiments indicate the effectiveness of our proposed model **Rein*former***. Despite our promising improvement on trajectory stitching, sequence modeling still falls short compared to classical RL. In future work, we will focus on identifying and bridging the gap between classical RL algorithms and sequence modeling. Furthermore, we aim to investigate the scenarios where classical RL excels and where sequence models can truly shine, providing a more nuanced understanding of their respective strengths and applications.

Acknowledgements

This work was supported by the National Science and Technology Innovation 2030 - Major Project (Grant No. 2022ZD0208800), and NSFC General Program (Grant No. 62176215).

Impact Statement

Our research introduces a novel approach to offline reinforcement learning (RL) through the integration of max-return sequence modeling, which we term **Rein*former***. This work aims to address a critical gap in the current paradigm of sequence modeling by explicitly incorporating the core RL objective of maximizing returns. By doing so, we enhance the trajectory stitching capability, which is crucial for learning from sub-optimal data and improving the overall performance of RL algorithms.

The broader impact of our work extends to various domains where RL is applied, such as robotics, autonomous systems, and decision-making processes. The **Rein*former*** algorithm the potential to lead to more efficient and effective learning algorithms, which could accelerate the development and deployment of autonomous technologies. This, in turn, could have significant societal consequences, including advancements in healthcare, transportation, and environmental management, where the ability to make optimal decisions is paramount.

From an ethical standpoint, our work emphasizes the importance of aligning RL algorithms with their intended objectives, ensuring that the pursuit of maximizing returns does not lead to unintended consequences. This is particularly relevant as AI systems become more integrated into critical systems where ethical considerations are paramount.

As we move forward, it is imperative to consider the societal implications of deploying RL algorithms that are more capable of learning from data. This includes ensuring that these systems are transparent, fair, and accountable, and that they do not perpetuate biases or lead to negative outcomes for vulnerable populations.

In conclusion, while our work contributes to the advancement of the field of Machine Learning, particularly in the area of RL, we recognize the need for ongoing discussions around the ethical use and societal impact of these technologies. We encourage the community to engage in these conversations and to develop guidelines that will ensure the responsible application of our findings.

References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A., and Bellemare, M. G. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

Aigner, D. J., Amemiya, T., and Poirier, D. J. On the estimation of production frontiers: maximum likelihood estimation of the parameters of a discontinuous density function. *International economic review*, pp. 377–396, 1976.

Bai, C., Wang, L., Yang, Z., Deng, Z., Garg, A., Liu, P., and Wang, Z. Pessimistic bootstrapping for uncertainty-driven offline reinforcement learning. *arXiv preprint arXiv:2202.11566*, 2022.

Brandfonbrener, D., Whitney, W., Ranganath, R., and Bruna, J. Offline rl without off-policy evaluation. *Advances in neural information processing systems*, 34:4933–4946, 2021.

Brandfonbrener, D., Bietti, A., Buckman, J., Laroche, R., and Bruna, J. When does return-conditioned supervised learning work for offline reinforcement learning? *Advances in Neural Information Processing Systems*, 35: 1542–1553, 2022.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

David, S. B., Zimerman, I., Nachmani, E., and Wolf, L. Decision s4: Efficient sequence-based rl via state spaces layers. In *The Eleventh International Conference on Learning Representations*, 2022.

Degris, T., White, M., and Sutton, R. S. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Fujimoto, S. and Gu, S. S. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

Fujimoto, S., Meger, D., and Precup, D. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pp. 2052–2062. PMLR, 2019.

Gulcehre, C., Colmenarejo, S. G., Wang, Z., Sygnowski, J., Paine, T., Zolna, K., Chen, Y., Hoffman, M., Pascanu, R., and de Freitas, N. Regularized behavior value estimation. *arXiv preprint arXiv:2103.09575*, 2021.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Kim, J., Lee, S., Kim, W., and Sung, Y. Decision convformer: Local filtering in metaformer is sufficient for decision making. *arXiv preprint arXiv:2310.03022*, 2023.

Konda, V. and Tsitsiklis, J. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

Kostrikov, I., Fergus, R., Tompson, J., and Nachum, O. Offline reinforcement learning with fisher divergence critic regularization. In *International Conference on Machine Learning*, pp. 5774–5783. PMLR, 2021a.

Kostrikov, I., Nair, A., and Levine, S. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021b.

Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32, 2019.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 1179–1191, 2020.

Kumar, A., Agarwal, R., Geng, X., Tucker, G., and Levine, S. Offline q-learning on diverse multi-task data both scales and generalizes. *arXiv preprint arXiv:2211.15144*, 2022.

Lee, K.-H., Nachum, O., Yang, M. S., Lee, L., Freeman, D., Guadarrama, S., Fischer, I., Xu, W., Jang, E., Michalewski, H., et al. Multi-game decision transformers. *Advances in Neural Information Processing Systems*, 35: 27921–27936, 2022.

Levine, S., Kumar, A., Tucker, G., and Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Liu, J., Zhang, H., and Wang, D. Dara: Dynamics-aware reward augmentation in offline reinforcement learning. *arXiv preprint arXiv:2203.06662*, 2022.

Liu, J., He, L., Kang, Y., Zhuang, Z., Wang, D., and Xu, H. Ceil: Generalized contextual imitation learning. *Advances in Neural Information Processing Systems*, 36: 75491–75516, 2023.

Liu, J., Guo, X., Zhuang, Z., and Wang, D. Didi: Diffusion-guided diversity for offline behavioral generation. *arXiv preprint arXiv:2405.14790*, 2024a.

Liu, J., Zhang, H., Zhuang, Z., Kang, Y., Wang, D., and Wang, B. Design from policies: Conservative test-time adaptation for offline policy optimization. *Advances in Neural Information Processing Systems*, 36, 2024b.

Liu, J., Zhang, Z., Wei, Z., Zhuang, Z., Kang, Y., Gai, S., and Wang, D. Beyond ood state actions: Supported cross-domain offline reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 13945–13953, 2024c.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Newey, W. K. and Powell, J. L. Asymmetric least squares estimation and testing. *Econometrica: Journal of the Econometric Society*, pp. 819–847, 1987.

Pomerleau, D. A. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

Schmidhuber, J. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sobotka, F. and Kneib, T. Geoadditive expectile regression. *Computational Statistics & Data Analysis*, 56(4):755–767, 2012.

Srivastava, R. K., Shyam, P., Mutz, F., Jaśkowski, W., and Schmidhuber, J. Training agents using upside-down reinforcement learning. *arXiv preprint arXiv:1912.02877*, 2019.

Sutton, R. S., Barto, A. G., et al. Introduction to reinforcement learning. 1998.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Tarasov, D., Nikulin, A., Akimov, D., Kurenkov, V., and Kolesnikov, S. CORL: Research-oriented deep offline reinforcement learning library. In *3rd Offline RL Workshop: Offline RL as a "Launchpad"*, 2022. URL https://openreview.net/forum?id=SyAS49bBcv.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, Z., Novikov, A., Zolna, K., Merel, J. S., Springenberg, J. T., Reed, S. E., Shahriari, B., Siegel, N., Gulcehre, C., Heess, N., et al. Critic regularized regression. *Advances in Neural Information Processing Systems*, 33: 7768–7778, 2020.

Watkins, C. J. and Dayan, P. Q-learning. *Machine learning*, 8:279–292, 1992.

Wu, Y., Tucker, G., and Nachum, O. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Wu, Y.-H., Wang, X., and Hamaya, M. Elastic decision transformer. *arXiv preprint arXiv:2307.02484*, 2023.

Xie, Z., Lin, Z., Ye, D., Fu, Q., Wei, Y., and Li, S. Future-conditioned unsupervised pretraining for decision transformer. In *International Conference on Machine Learning*, pp. 38187–38203. PMLR, 2023.

Yamagata, T., Khalil, A., and Santos-Rodriguez, R. Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline rl. In *International Conference on Machine Learning*, pp. 38989–39007. PMLR, 2023.

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C.-J. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

Zheng, Q., Zhang, A., and Grover, A. Online decision transformer. In *international conference on machine learning*, pp. 27042–27059. PMLR, 2022.

Zhuang, Z., Lei, K., Liu, J., Wang, D., and Guo, Y. Behavior proximal policy optimization. *arXiv preprint arXiv:2302.11312*, 2023.

A. Baseline Results

Existing sequence modeling algorithms have only been evaluated on the `Gym-v2` dataset, neglecting other more complex and challenging datasets (`Maze2d-v1`, `Kitchen-v0` and `Antmaze-v2`). To provide a more comprehensive comparison and evaluation of **Rein*former***, we have reproduced several baseline methods using their offical code. Subsequently, we will clarify which results stem from the original works or third-party reproductions, and which are reproduced by ourselves. Table 1 contains **last** results on dataset `Gym-v2`, `Maze2d-v1` and `Kitchen-v0`.

- `Gym-v2`: All the results of Reinforcement Learning baselines including BC, CQL and IQL come from the IQL original paper (Kostrikov et al., 2021b). The results of Decision Transformer (DT) are from the DT original paper (Chen et al., 2021). For ODT, EDT and QDT, the results are sourced from their respective original papers (Zheng et al., 2022; Wu et al., 2023; Yamagata et al., 2023) while the results on dataset `medium-expert` are all absent. We reproduce the EDT which considers trajectory stitching using its official code `https://github.com/kristery/Elastic-DT`.
- `Maze2d-v1`: The results of BC, CQL, IQL and DT are from the open source library Clean Offline Reinforcement Learning (CORL) `https://github.com/tinkoff-ai/CORL`. CORL provides high-quality and easy-to-follow single-file implementations of state-of-the-art offline reinforcement learning algorithms and each implementation is backed by a research-friendly codebase (Tarasov et al., 2022). For QDT, the results come from its original paper (Yamagata et al., 2023) without discounted returns-to-go introduced. The result of EDT is still reproduced.
- `Kitchen-v0`: The results of BC, CQL and IQL come from the IQL original paper (Kostrikov et al., 2021b) and we reproduce the EDT.

Table 2 contains the **best** results on dataset `Antmaze-v2`. The reason for selecting the best result rather than the last one is due to the instability in the training of **Rein*former***. Additionally, our algorithm's performance on `Antmaze-large-diverse` and `Antmaze-large-play` is poor, with the agent completing the tasks only 1 or 2 times out of 100 evaluations. It is challenging to ascertain whether these results are attributable to the randomness induced by hyperparameter tuning or seed selection. So we only report the results on other four datasets.

- `Antmaze-v2`: The results of BC, IQL and DT are extracted from CORL library (Tarasov et al., 2022). The other results are reproduced by ourselves using its their respective official code, including ODT `https://github.com/facebookresearch/online-dt` and EDT `https://github.com/kristery/Elastic-DT`.

During the process of reproducing EDT, we observe that EDT is highly sensitive to the initial returns-to-go. This indicates that, despite incorporating a loss function aimed at maximizing returns, EDT still operates with naively maximizing returns. Additionally, we are not fully acquainted with the optimal selection of initial returns-to-go on `Maze2d-v1`, `Kitchen-v0` and `Antmaze-v2`. To address this, we have uniformly adopted the maximum return observed in the dataset. However, we are uncertain whether this choice may introduce any adverse effects on the experimental outcomes of EDT. This indirectly highlights one of the advantages of **Rein*former***, which is its ability to adaptively select the returns-to-go without the need for manual design.

B. Experiment Details and Hyperparameters

Our **Rein*former*** implementation draws inspiration from and references the following four repositories:

- online-dt: `https://github.com/facebookresearch/online-dt`;
- Elatic-DT: `https://github.com/kristery/Elastic-DT`;
- min-decision-transformer: `https://github.com/nikhilbarhate99/min-decision-transformer`;
- decision-transformer: `https://github.com/kzl/decision-transformer`.

The state tokens, return tokens and action tokens are first processed by different linear layers. Then these tokens are fed into the decoder layer to obtain the embedding. Here the decoder layer is a lightweight implementation from "min-decision-transformer". The context length for the decoder layer is denoted as K. We use the LAMB (You et al., 2019) optimizer with to optimize the model with action loss and returns-to-go loss. The hyperparameter of returns loss is denoted as m.

B.1. Hyperparameter m

The hyperparameter m is crucially related to the returns-to-go loss and is one of our primary focuses for tuning. We explore values within the range of $m = [0.7, 0.9, 0.99, 0.999]$. When $m = 0.5$, the expectile loss function will degenerate into MSE loss, which means the model is unable to output a maximized returns-to-go. So we do not take $m = 0.5$ into consideration. We observe that performance is generally lower at $m = 0.7$ compared to others. Only `hopper-medium` and `hopper-medium-replay` adopt the parameter $m = 0.999$ while $m = 0.9$ and $m = 0.99$ are generally better than $m = 0.999$ on other datasets. The detailed hyperparameter selection of m is summarized in the following table:

Table 3. Hyperparameters m of returns loss on different datasets.

Dataset	m		
`halfcheetah-medium`	0.9	`maze2d-umaze`	0.99
`halfcheetah-medium-replay`	0.9	`maze2d-medium`	0.99
`halfcheetah-medium-expert`	0.9	`maze2d-large`	0.99
`hopper-medium`	0.999	`kitchen-complete`	0.99
`hopper-medium-replay`	0.999	`kitchen-partial`	0.9
`hopper-medium-expert`	0.9	`kitchen-mixed`	0.9
`walker2d-medium`	0.9	`Antmaze-umaze`	0.9
`walker2d-medium-replay`	0.99	`Antmaze-umaze-diverse`	0.99
`walker2d-medium-expert`	0.99	`Antmaze-medium-play`	0.99
		`Antmaze-medium-diverse`	0.99

B.2. Context Length K

The context length K is another key hyperparameter for sequence modeling, and we conduct a parameter search across the values $K = [2, 5, 10, 20]$. The maximum value is 20 because the default context length for DT (Chen et al., 2021) is 20. The minimum is 2, which corresponds to the shortest sequence length (setting $K = 1$ would no longer constitute sequence modeling). Overall, we found that $K = 20$ leads to more stable learning and better performance on high quality dataset (such as `Gym-medium-expert` and `Kitchen`). Conversely, a smaller context length is preferable on low quality dataset (such as `Gym-medium/medium-replay` and `Antmaze`). The parameter K has been summarized as follows:

Table 4. Context length K on different datasets.

Dataset	K		
`halfcheetah-medium`	5	`maze2d-umaze`	20
`halfcheetah-medium-replay`	5	`maze2d-medium`	10
`halfcheetah-medium-expert`	20	`maze2d-large`	10
`hopper-medium`	5	`kitchen-complete`	20
`hopper-medium-replay`	5	`kitchen-partial`	20
`hopper-medium-expert`	20	`kitchen-mixed`	20
`walker2d-medium`	5	`Antmaze-umaze`	2
`walker2d-medium-replay`	2	`Antmaze-umaze-diverse`	2
`walker2d-medium-expert`	20	`Antmaze-medium-play`	3
		`Antmaze-medium-diverse`	2

B.3. Training Steps and Learning Rate

The default number of training steps is 50000, with a learning rate of 0.0001. With these default settings, if the training score continues to rise, we would consider increasing the number of training steps or doubling the learning rate. For some datasets, 50000 steps may cause overfitting and less training steps are better. The training steps are presented in Table 5 and learning rate is summarized in Table 6. We evaluate the policy every 5000 steps to obtain a normalized score. For each seed, this normalized score is calculated as the average returns of 100 trajectories except for 10 trajectories on `Gym-v2` datasets.

Table 5. The training steps on different datasets.

Dataset	Training Steps	maze2d-umaze	50000
halfcheetah-medium	50000	maze2d-medium	35000
halfcheetah-medium-replay	50000	maze2d-large	45000
halfcheetah-medium-expert	50000	kitchen-complete	100000
hopper-medium	30000	kitchen-partial	90000
hopper-medium-replay	300000	kitchen-mixed	50000
hopper-medium-expert	100000	Antmaze-umaze	50000
walker2d-medium	15000	Antmaze-umaze-diverse	50000
walker2d-medium-replay	80000	Antmaze-medium-play	100000
walker2d-medium-expert	50000	Antmaze-medium-diverse	100000

Table 6. The learning rate on different datasets.

Learning Rate	Dataset
0.0008	Antmaze-medium-play;
	1) walker2d-medium-replay;
0.0004	2) maze2d-medium; 3) maze2d-large;
	4) Antmaze-medium-diverse;
0.0001	Other datasets.

B.4. Network Architecture

The default values for the number of decoder layer, attention heads and hidden dimension are 4, 8 and 256, respectively. These parameters are usually fixed. When we observe an initial increase followed by a decrease in the training curve, we infer overfitting and reduce the number of layers. On the contrary, if the training curve consistently rises without a clear convergence trend, we would attempt to increase the number of layers. As for the number of attention heads, only Antmaze-medium-diverse is 4.

Table 7. The number of encoder layers and attention heads on different datasets.

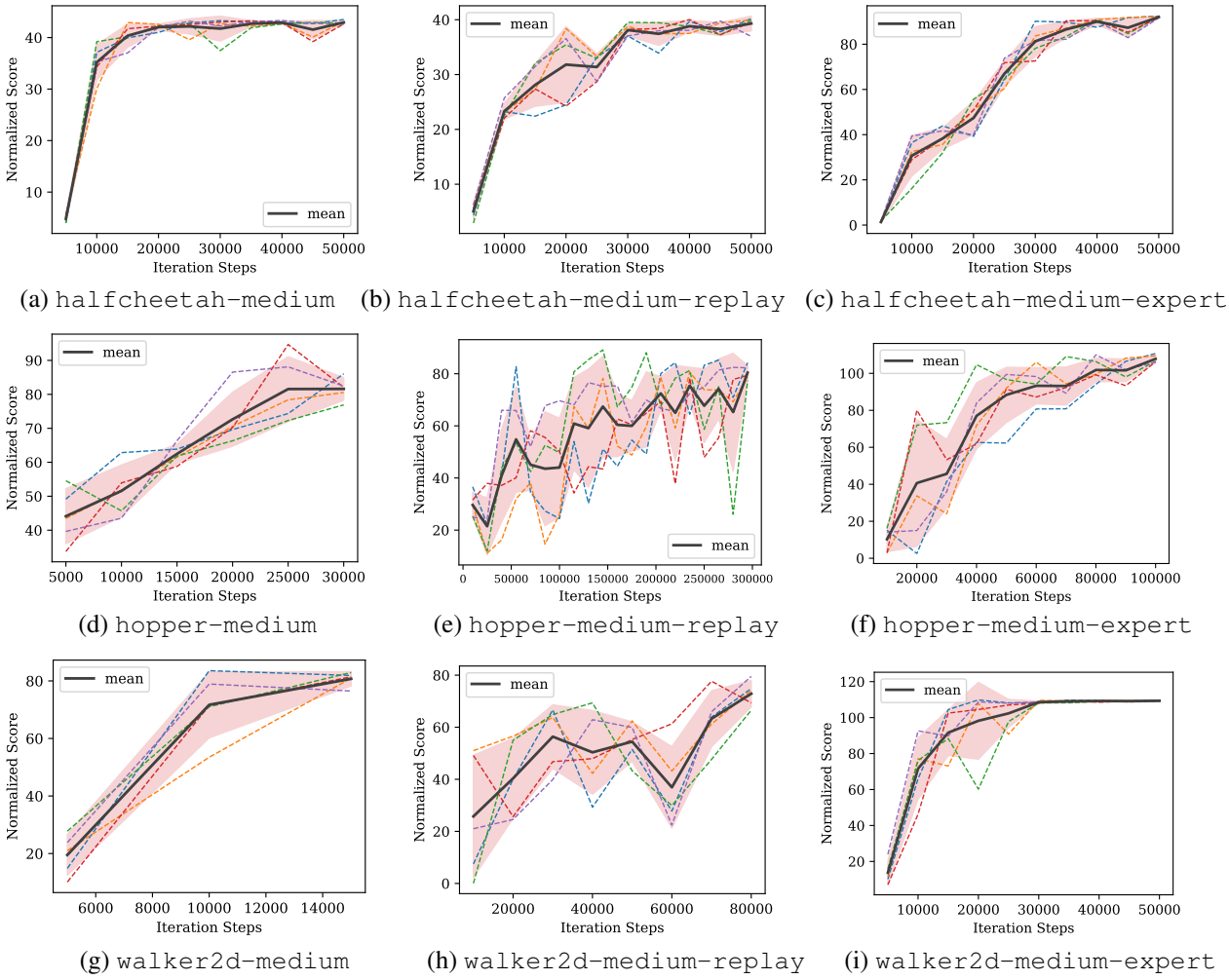
Hyperparameters	Values	Dataset
Encoder Layers	3	maze2d-umaze; antmaze-umaze; antmaze-umaze-diverse;
	4	Gym-v2;
	5	maze2d-medium; maze2d-large; Kitchen-v0; Antmaze-medium-diverse;
	6	Antmaze-medium-play.
Attention Heads	4	Antmaze-medium-diverse;
	8	Other datasets.

C. Training Curves

We exhibit the training curves on five seeds. The black line represents the mean of these five seeds and the red shaded area represents the variance.

C.1. Gym-v2

The training curves on nine datasets from Gym-v2 are plotted in Figure 8. Among these nine datasets, hopper-medium-replay exhibits severe training fluctuations, while walker2d-medium-replay shows slight fluctuation. The remaining datasets are notably stable, yielding satisfactory results without deliberate hyperparameter tuning.



(a) halfcheetah-medium (b) halfcheetah-medium-replay (c) halfcheetah-medium-expert

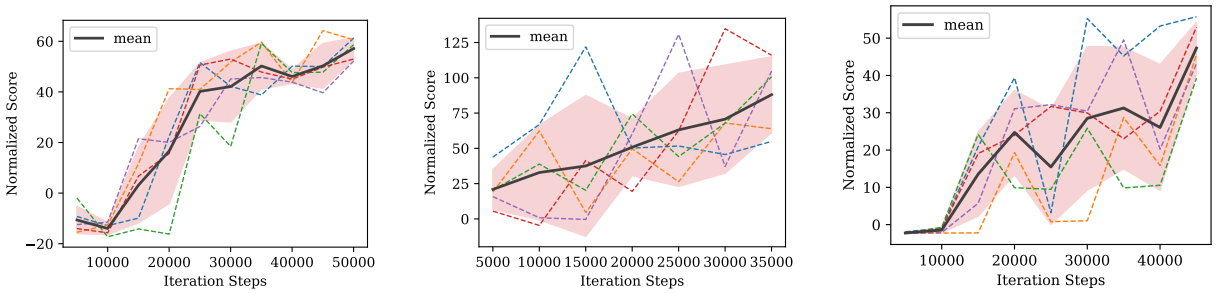
(d) hopper-medium (e) hopper-medium-replay (f) hopper-medium-expert

(g) walker2d-medium (h) walker2d-medium-replay (i) walker2d-medium-expert

Figure 8. The training curves on Gym-v2.

C.2. Maze2d-v1

The training curves on three datasets from Maze2d-v1 are plotted in Figure 9. The training curve of maze2d-umaze is relatively stable. The variance on dataset maze2d-medium is very high and the training process also suffers from severe training fluctuations. Sometimes, the score can even approach to 125. Dataset maze2d-medium also fluctuates a little.

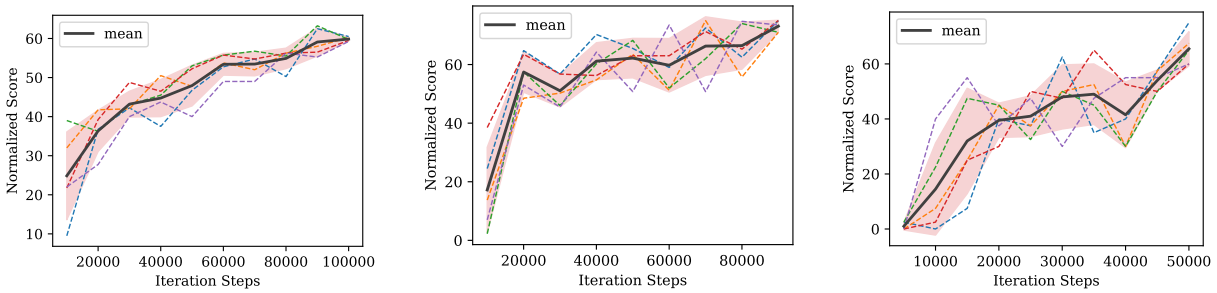


(a) maze2d-umaze (b) maze2d-medium (c) maze2d-large

Figure 9. The training curves on Maze2d-v1.

C.3. `Kitchen-v0`

The training curves on three datasets from `Kitchen-v0` are plotted in Figure 9. Overall, the training curves are remarkably stable, and the long context length K is crucial for the stability of learning.



(a) `kitchen-complete` (b) `kitchen-partial` (c) `kitchen-mixed`

Figure 10. The training curves on `Kitchen-v0`.

C.4. `Antmaze-v2`

Since we report the best score during training rather than the last score, we do not provide training curves on `Antmaze`. Here, we would like to emphasize the reward modification. `Antmaze` contains datasets with sparse rewards, where only 1 indicates the reach of the goal while 0 is not. In order to enhance the reward signal, we multiply the reward by 100. However, we found that this modification leads to the occurrence of NaN values on dataset `Antmaze-umaze-diverse`. Besides, the original reward also occur the NaN values. So we modify the reward by adding another 1, that is, $\hat{r} = 100 \times r + 1$. We summarize the results of different reward modification in Table 8.

Table 8. The normalized best score on `Antmaze-v2` dataset with different reward modification.

`Antmaze`	$\hat{r} = 100 \times r$	$\hat{r} = 100 \times r + 1$
`umaze`	**85**±**3.87**	84.4±2.7
`umaze-diverse`	NaN	**65.8**±**4.1**
`medium-play`	11.4±3.78	**13.2**±**6.1**
`medium-diverse`	7.2±2.17	**10.6**±**6.9**