Towards Long-delayed Sparsity: Learning a Better Transformer through Reward Redistribution

Tianchen Zhu[1,2,4] , **Yue Qiu**[1,2] , **Haoyi Zhou**[1,3] and **Jianxin Li**[1,2]
[1]Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University
[2]School of Computer Science and Engineering, Beihang University
[3]School of Software, Beihang University
[4]Shenyuan Honors College, Beihang University
{zhutc,qiuyue,zhouhy,lijx}@act.buaa.edu.cn

Abstract

Recently, Decision Transformer (DT) pioneered the offline RL into a contextual conditional sequence modeling paradigm, which leverages self-attended autoregression to learn from global target rewards, states, and actions. However, many applications have a severe delay of the above signals, such as the agent can only obtain a reward signal at the end of each trajectory. This delay causes an unwanted bias cumulating in autoregressive learning global signals. In this paper, we focused its virtual example on episodic reinforcement learning with trajectory feedback. We propose a new reward redistribution algorithm for learning parameterized reward functions, and it decomposes the long-delayed reward onto each timestep. To improve the redistributing's adaptation ability, we formulate the previous decomposition as a bi-level optimization problem for global optimal. We extensively evaluate the proposed method on various benchmarks and demonstrate an overwhelming performance improvement under long-delayed settings.

1 Introduction

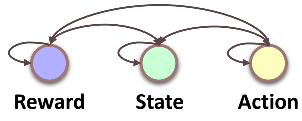
The large-scale generative models of sequence modeling have achieved great success with widespread application in NLP [Devlin *et al.*, 2019], CV [Dosovitskiy *et al.*, 2021; Yang *et al.*, 2023], time-series forecasting [Zhou *et al.*, 2021]. They have evolved into a common paradigm and dominant approach in many machine learning domains.

Recently, Decision Transformer (DT) [Chen *et al.*, 2021] pioneered modeling reinforcement learning (RL) as a sequence generation problem, intending to generate a sequence of actions that, when implemented in an environment, will yield a series of high returns. In contrast to traditional online RL approaches, this new offline RL paradigm, which learns from a fixed-size dataset by modeling conditional sequences, can effectively overcome the difficulty of collecting data samples in many real-world scenarios. DT proposes to model RL trajectories using Transformer [Vaswani *et al.*, 2017], and it attempts to leverage self-attended autoregression to learn how to make optimal actions by modeling and to extract six types of **heterogeneous dependencies** between states, actions, and target rewards in unstructured trajectory sequences (as shown in Figure 1(a)). DT can effectively mitigate the instability of the bootstrap process and short-sightedness of reward propagation during training traditional RL algorithms (e.g., TD learning [Sutton and Barto, 1998]), and are exceptionally superior in capturing long-term dependencies.

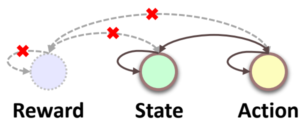
The DT model generally outperforms traditional offline RL methods such as Behavioral Cloning (BC) [Torabi *et al.*, 2018] due to its ability to capture multiple heterogeneous dependencies among target rewards, states, and actions. DT has been successfully utilized in environments with dense or slightly sparse rewards, benefiting from its modeling of global dependencies for target reward signals, which is similar to Q-value learning in online RL. However, a more challenging issue arises in applications where reward signals are severely delayed, and the agent can only receive a reward signal at the end of each trajectory. This is the case in scenarios like mazes and machine control, where rewards are only available in the final state. Unlike the aforementioned sparse rewards, the problem of long-delayed rewards involves not only the scarcity of reward signals but also the temporal structure. Directly applying the DT model to such long-delayed reward environments would lead to significant challenges.

The problem can be explicitly expressed as the accumulation of unwanted biases in autoregressive learning global signals due to reward delays. This bias can be specified as a bias in the attention distribution and is transmitted layer by layer in the neural network, thus triggering the defect of model performance degradation and reducing the effectiveness of the decision model. More specifically, in scenarios with long-delayed rewards, the semantic dependencies between reward signals and timesteps become significantly inconsistent in the collected offline trajectories. Since the reward is only obtained at the last timestep, all target rewards within the trajectory, except for the last timestep, are identical. Consequently, the majority of reward signals in this setup become redundant and may even weaken or mislead the classical self-attention model in extracting complete heterogeneous dependencies and learning unbiased attention distributions. These unwanted biases tend to degrade the DT model into a behavior cloning model, i.e., modeling only three types

The corresponding author is Jianxin Li. The source code is available at https://github.com/catezi/DTRD.



(a) Complete heterogeneous dependencies.



(b) Incomplete heterogeneous dependencies.

Figure 1: Heterogeneous dependencies captured by the DT model among target rewards, states, and actions at the attention correlation level. The left side represents the dense reward setting, while the right side represents the sparse reward setting.

of heterogeneous dependencies between states and actions while disregarding target rewards (as shown in Figure 1(b)).

Therefore, this paper focuses on providing a virtual example of episodic reinforcement learning with trajectory feedback. To address the above problem, we aim to build a new reward redistribution algorithm named **DTRD** for learning parameterized reward functions, and it decomposes the long-delayed reward onto each timestep. To enhance the adaptability of the redistribution process, we formulate the previous decomposition as a bi-level optimization problem to automatically evaluate the effectiveness of modeling the reward function. This evaluation involves assessing the impact of redistributed rewards on improving the performance of the policy model. At the lower level, we optimize a policy model to extract expert policies embedded in the offline trajectories. At the upper level, we optimize a parameterized reward shaping function and aim to maximize the decision performance of the lower-level policy model on specific tasks while learning an appropriate way to redistribute rewards. Our algorithm also contributes to improving the performance of DT in general sparse reward problems. In conclusion, our proposed algorithm automates the search for optimal and reasonable reward decomposition functions by establishing a bi-level optimization framework. This reward redistribution facilitates optimal policy optimization. Our algorithm significantly enhances the effectiveness of DT models in long-delayed reward environments and improves their generalization ability. The contributions of this paper can be summarized as follows:

- Firstly, we reveal the problem of unwanted biases accumulation under long-delayed settings for models that learn global signals autoregressively for the first time.
- Secondly, we propose an adaptive reward redistribution method based on bi-level optimization. It redefines a more direct reward redistribution optimization objective in terms of the validation loss of the policy model.
- Thirdly, we provide formal results for computing the parameter gradient of the policy validation loss with respect to the reward function. This enables the decoupling of policy optimization and reward redistribution, leading to accelerated training.
- Finally, we extensively evaluate the proposed method DTRD on various benchmarks, including Atari, Minigrid, D4RL, etc., and demonstrate an overwhelming performance improvement in long-delayed settings.

2 Background

2.1 Setup and Notation

We are interested in decision transformer (DT) [Chen *et al.*, 2021] in episodic task settings, wherein an agent can only obtain one reward feedback at the end of each trajectory. Let τ denote a trajectory, and let $|\tau|$ denote its length. The return-to-go (rtg) $g_t = \sum_{t'=t}^{|\tau|} r_{t'}$ of a trajectory τ at timestep t denotes the sum of future rewards from that timestep. Let $\mathbf{G} = (g_1, ..., g_{|\tau|})$, $\mathbf{S} = (s_1, ..., s_{|\tau|})$ and $\mathbf{A} = (a_1, ..., a_{|\tau|})$ denote the sequence of rtgs, states, actions of τ, respectively.

2.2 Decision Transformer

Decision Transformer (DT) processes a trajectory τ as a sequence of 3 types of input tokens: rtgs, states, and actions: $(g_1, s_1, a_1, g_2, s_2, a_2, ..., g_{|\tau|}, s_{|\tau|}, a_{|\tau|})$, where the initial rtg g_1 is equal to the accumulated reward of the whole trajectory (trajectory feedback). At timestep t, DT uses the tokens from the latest k timesteps to generate an action a_t, where k is a hyperparameter and denotes the context length for the Transformer model. It is worth noting that the context length during the evaluation can be shorter than the context length used for training. DT learns a deterministic policy $\pi(a_t|\mathbf{g}_{-k,t}, \mathbf{s}_{-k,t}, \mathbf{a}_{-k,t-1})$, where $\mathbf{s}_{-k,t}$ denotes the sequence of k past states and similarly for $g_{-k,t}$ and $a_{-k,t-1}$. Therefore, DT is essentially a k-order autoregressive model. DT parameterized the policy in particular using a GPT architecture [Radford and Narasimhan, 2018], which employs a causal mask to enforce such an autoregressive structure in the predicted action sequence. The data distribution $\mathcal{T}$ is assumed to generate length-K rtg $\mathbf{g}$, action $\mathbf{a}$ and state $\mathbf{s}$ subsequences from the same trajectory. This allows us to easily present our approach's training objective, and the policy is trained to predict action tokens using the standard L2-loss.

$$L_{train} = \mathbb{E}_{(\mathbf{g},\mathbf{s},\mathbf{a})\sim\mathcal{T}}\left[\frac{1}{k}\sum\nolimits_{t=1}^{k}(\mathbf{a}_t - \pi(\mathbf{g}_{-k,t}, \mathbf{s}_{-k,t}))^2\right]. \tag{1}$$

2.3 Episodic RL with Trajectory Feedback

The environment model in typical RL scenarios is usually formulated by a finite Markov decision process (MDP), which is defined as a 5-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the spaces of environment states and agent actions [Bellman, 1966]. Moreover, $R(s, a)$ denotes the environment transition and reward function [Ren *et al.*, 2022]. Furthermore, $\mathcal{M}$ has transition distribution $P(s'|s, a) = p(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)$ and reward function $R(s, a)$ conditioned on state-actions and a discount factor $\gamma \in [0, 1]$. In this paper, we consider the episodic RL setting with trajectory feedback, where the agent can only obtain one reward feedback at the end of each trajectory. In other words, before reaching the final state s_T, rewards $R(s_t, a_t) = 0$ for all $t < T$, where T denotes the finite horizon length. In many episodic tasks, the terminal states can be predefined, or the length of the trajectories is limited. Let $\tau = \{s_t, a_t\}_{t=1}^{T}$ denote an agent trajectory that includes all experienced states and actions within an episode. Then the episode reward function $R_{ep}(\tau)$ can be defined on the trajectory space, representing the overall performance of trajectory τ. In fact, the episodic reward setting is a particular form of partially observable Markov decision process (POMDP), where the reward function is non-Markovian. The worst-case may lead the agent to traverse the entire exponential-scale trajectory space to represent the episodic reward function. A common structural assumption is the existence of an underlying Markovian reward function $\hat{R}_{step}(s,a)$ that approximates the episodic reward $R_{ep}(\tau)$ by a sum-form decomposition

$$R_{ep}(\tau) = R_{step}(s_T, a_T) \approx \hat{R}_{ep}(\tau) = \sum\nolimits_{t=1}^{T} \hat{R}_{step}(s_t, a_t). \tag{2}$$

This form is frequently employed in many studies [Ren *et al.*, 2022; Efroni *et al.*, 2021; Liu *et al.*, 2019] on episodic RL with trajectory feedback.

2.4 Reward Redistribution

The goal of reward redistribution is to build a shaping reward function $\hat{R}_{step}(s_t, a_t)$ that helps convert the episodic-reward problem to the standard dense-reward setting. A reward redistribution procedure redistributes the trajectory feedback variable $R_{ep}(\tau)$ along the sequence for each trajectory τ. Formally, the proxy rewards $\hat{R}_{step}(s_t, a_t)$ form a sum-decomposable reward function $\hat{R}_{ep}(\tau) = \sum_{t=1}^{T} \hat{R}_{step}(s_t, a_t)$ that is expected to have a high correlation to the environmental delayed reward $\hat{R}_{ep}(\tau)$.

3 Methodology

In this section, we present DTRD , an improved adaptive reward redistribution method that leverages the optimization performance of a policy model as an explicit objective to guide the learning of the reward function. We formulate reward redistribution and policy optimization as a bi-level optimization problem. To accelerate model training, we decouple interacting objective functions of upper and lower levels using a single-step gradient approximation.

3.1 Motivation

A common approach to address the issue of sparse or delayed rewards in traditional online scenarios is through shaping reward signals. The main objective is to transform sparse or delayed rewards into dense Markovian reward functions that better represent the intended goals indicated by the environmental feedback. This approach typically involves reward modeling and policy optimization, where an agent iteratively refines its reward function by interacting with the online environment [Li *et al.*, 2023; Fu *et al.*, 2021]. For instance, Sorg *et al.* treated reward function design as an online problem, employing a trial-and-error loop in reinforcement learning [Sorg *et al.*, 2010; Arjona-Medina *et al.*, 2019; Rajeswaran *et al.*, 2020]. This paradigm can also be applied to offline scenarios like DT to mitigate model degradation in long-delayed reward environments. The general idea is to develop a two-stage pipeline comprising reward function estimation and policy model optimization. Firstly, we determine the "locally optimal" form of the reward function by utilizing the relationship between rewards and state-action pairs in offline data, often employing methods like least-squares estimation. Then, we adjust and align the reward signals in the original data based on the approximated reward function to improve the optimization of the policy model.

The existing two-stage optimization pipeline has two inherent issues. Firstly, its optimization objective of reward function lacks directness, making it susceptible to local optima. Secondly, the two-stage process amplifies the negative impact of the locally optimal reward function, leading to error accumulation and significant deviations from expected results in policy optimization. To overcome these limitations and achieve a "globally optimal" solution for the reward function, a more effective approach is to integrate reward modeling and policy optimization into a cohesive "adaptive" optimization process. This adaptivity means automatically guiding the modeling of the reward function by evaluating the impact of policy optimization. To achieve this, we propose a bi-level optimization framework that provides a more direct and meaningful objective function for searching and optimizing the reward function.

3.2 Parameterized Shaping Reward Function

Given an offline behavior trajectory set $\mathcal{S} = \{\tau_1, \tau_2, ..., \tau_n\}$, our goal is to redistribute the trajectory reward onto each timestep. We need to reshape the reward function and ensure that the proxy rewards can help learn a better policy. In other words, we need to find the suitable shaping reward function to correct rtgs in the trajectories so that the policy learned from these modified trajectories is globally optimal. So we first introduce a proxy reward function f_ϕ, then the new form of the reward $\hat{r}_t$ at time step t can be written as

$$\hat{r}_t = \hat{R}_{step}(s_t, a_t) = f_\phi(s_t, a_t), \tag{3}$$

$$\text{s.t. } R_{ep}(\tau) = \sum\nolimits_{t=1}^{|\tau|} \hat{r}_t, \tag{4}$$

where $s_t, a_t \in \tau$, and $f_\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the shaping reward function that assigns a modified reward value to each state-action pair and is parameterized by ϕ. Since the DT model takes return-to-go (rtg) g_t instead of reward r_t as input, we need to further calculate shaping rtgs based on shaping rewards. Specifically, the shaping rtg $\hat{g}_t$ can be modeled as

$$\hat{g}_t = g_t - \sum\nolimits_{i=1}^{t-1} \hat{r}_i, \tag{5}$$

$$\text{s.t. } R_{ep}(\tau) = \sum\nolimits_{t=1}^{|\tau|} \hat{r}_t. \tag{6}$$

3.3 Bi-level Optimization

To learn the optimal shaping reward function, Efroni *et al.* provided a basic idea to train a regression model that decomposed the trajectory reward into the sum of the agent rewards at each step [Efroni *et al.*, 2021; Ren *et al.*, 2022]. However, These methods have limitations as they are unidirectional and rely solely on regressing the reward function for agent training, which can lead to overfitting and lacks automatic adjustment for longer-delayed settings. To address these issues, we propose an end-to-end bi-directional bi-level optimization

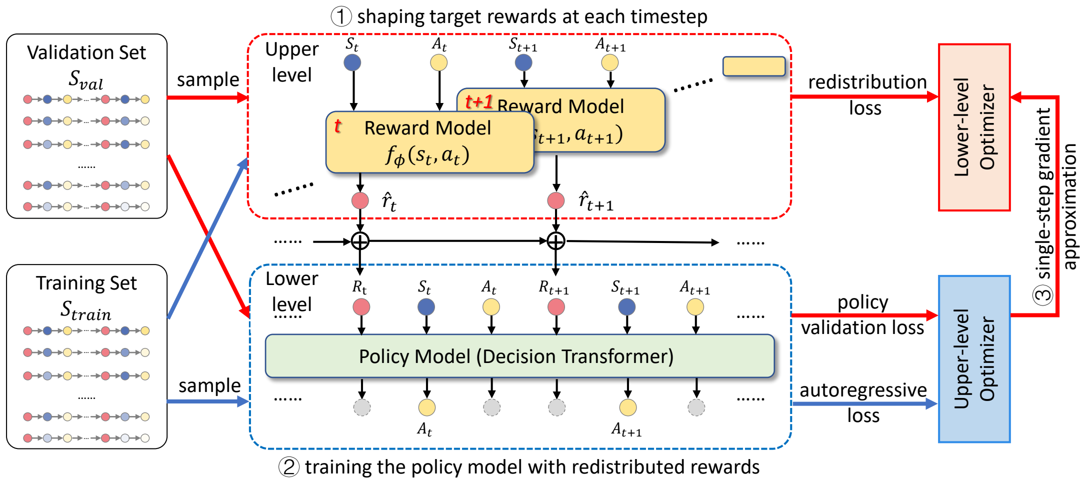


Figure 2: An illustration of the DTRD architecture. We divided offline trajectories into training and validation sets. DTRD consists of the following steps: (1) Shaping target rewards: sampling trajectories from training and validation sets, and redistributing long-delayed rewards to each timestep by the reward model. (2) Policy model training for lower-level optimization: calculating shaping target rewards based on the redistributed rewards and training the policy model on the training set. (3) Single-step gradient approximation for upper-level optimization: computing the validation loss of the policy model with redistributed rewards on the validation set and computing the upper model's gradient by single-step gradient approximation. The redistribution loss represents the quadratic penalty term mentioned in Section 3.3.

framework that utilizes differentiable feedback to search the reward function. This framework is better suited for long-delayed settings and offers several advantages. Firstly, it allows the trained reward function to guide agent training while enabling the agent to adjust the reward function, forming a closed loop of bi-directional feedback and joint optimization. Secondly, it mitigates overfitting by optimizing the reward function through a higher-level optimization process. Inspired by [Liu *et al.*, 2018], our core idea is to **use a validation set to assess how the redistributed rewards enhance the policy model, thereby guiding the optimization of the reward function** (as shown in Fig. 2). Based on this, we divided all the trajectory data $\mathcal{S}$ into two categories: training set $\mathcal{S}_{train}$ and validation set $\mathcal{S}_{val}$. $\mathcal{S}_{train}$ is used for optimizing the policy model, and $\mathcal{S}_{val}$ is used for optimizing the reward function. The approach involved two tasks: the lower-level task focused on optimizing the policy model π_θ to learn the best policy on the training set, with the reward function corrected based on trajectory data. The upper-level task aimed to optimize the shaping reward function f_ϕ to find the reward function that maximizes the enhancement of the policy model. The goal was to ensure that the lower-level optimized policy model performs best on the validation set. We aimed to jointly learn the policy model π_θ and the shaping reward function f_ϕ. Unlike regression-based reward redistribution methods, where the optimization objective was the least-squares loss for reward decomposition, we aimed to optimize the shaping reward function by the task loss differentiably. Let $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ denote the training and the validation loss, respectively. Both losses are determined not by the policy model θ and the shaping reward function ϕ. The goal for searching suitable shaping reward function is to find ϕ^* that minimizes the validation loss $\mathcal{L}_{val}(\theta^*, \phi^*)$, where the policy θ^* associated with the proxy reward values are obtained by minimizing the training loss $\theta^* = \operatorname{argmin}_\theta\ \mathcal{L}_{train}(\theta, \phi^*)$.

Thus, the optimization of the policy model θ and the reward model ϕ forms a bi-level optimization problem, which can be formally defined as

$$\min_\phi\ \mathcal{L}_{val}(\theta^*(\phi), \phi), \tag{7}$$

$$\text{s.t.}\ \ R_{ep}(\tau) = \sum\nolimits_{t=1}^{|\tau|} \hat{r}_t,\ \forall \tau \in \mathcal{S}_{val}, \tag{8}$$

$$\text{s.t.}\ \ \theta^*(\phi) = \operatorname{argmin}_\theta\ \mathcal{L}_{train}(\theta, \phi), \tag{9}$$

where ϕ is the upper-level variable and θ is the lower-level variable, and $\hat{r}_t = f_\phi(s_t, a_t)$ is the shaping reward function formed by parameterizing ϕ. Eq. 8 is a constraint on reward redistribution that limits the sum of rewards to remain constant on the trajectories before and after the reward decomposition. We transformed this constrained bi-level optimization problem into an unconstrained optimization problem with minimalized auxiliary functions by defining the quadratic penalty function. Therefore, the objective function of the bi-level optimization problem can be rewritten as

$$\min_\phi\ \left[\mathcal{L}_{val}(\theta^*(\phi), \phi) + \lambda \sum_{\tau \in \mathcal{S}_{val}} \left(R_{ep}(\tau) - \sum_{t=1}^{|\tau|} \hat{r}_t\right)^2\right], \tag{10}$$

$$\text{s.t.}\ \ \theta^*(\phi) = \operatorname{argmin}_\theta\ \mathcal{L}_{train}(\theta, \phi), \tag{11}$$

where λ is a hyper-parameter to control the numerical scale balance between the penalty function and the original objective function when optimizing.

3.4 Gradient Approximation and Computation

We found that the upper-level optimization is tightly coupled to the lower-level optimization. However, accurately calculating the gradient of the upper-level reward model necessitates numerous iterations to determine the optimal policy at

Algorithm 1: Bi-level Optimization of Reward Redistribution

Input: Learning rates α_θ and α_ϕ for policy and reward optimization respectively, training dataset $\mathcal{S}_{train}$ and validation dataset $\mathcal{S}_{val}$.
Parameter: Parametrized θ for the policy model π_θ and parametrized ϕ for the reward model $f_\phi(s_t, a_t)$.

```
1: Initialize the policy parameter $\pi$ and the shaping reward function parameter $\phi$;
2: while not converged do
3:     $\phi \leftarrow \phi - \alpha_\phi \nabla_\phi [\mathcal{L}_{val}(\theta^*(\phi), \phi) + \lambda \sum_{\tau \in \mathcal{S}_{val}} \left( R_{ep}(\tau) - \sum_{t=1}^{|\tau|} \hat{r}_t \right)^2 ]$;
4:     $\theta \leftarrow \theta - \alpha_\theta \nabla_\theta \mathcal{L}_{train}(\theta, \phi)$;
5: return policy model $\pi_\theta$ and reward model $f_\phi(s_t, a_t)$.
```

the lower level, which can be prohibitively time-consuming. Inspired by [Liu *et al.*, 2018] and [Hu *et al.*, 2020], we devised an alternating optimization technique to approximate the gradient. This method efficiently separates the parameters of the two levels, leading to accelerated optimization. We proposed a simple approximation scheme as follows:

$$\theta^*(\phi) \approx \theta^{'}(\phi) = \theta - \alpha_\theta \nabla_\theta \mathcal{L}_{train}(\theta, \phi). \tag{12}$$

Thus the calculation of the upper-level gradient can be approximated as

$$\begin{aligned}
&\nabla_\phi \left[\mathcal{L}_{val}(\theta^*(\phi), \phi) + \lambda \sum\nolimits_{\tau \in \mathcal{S}_{val}} \left(R_{ep}(\tau) - \sum\nolimits_{t=1}^{|\tau|} \hat{r}_t \right)^2 \right] \\
\approx &\nabla_\phi \mathcal{L}_{val}(\theta^{'}(\phi), \phi) + \lambda \nabla_\phi \sum_{\tau \in \mathcal{S}_{val}} \left(R_{ep}(\tau) - \sum_{t=1}^{|\tau|} f_\phi(s_t, a_t) \right)^2
\end{aligned} \tag{13}$$

where θ denotes the current policy trained by the algorithm, and α_θ is the learning rate for a step of the lower-level policy optimization. The basic idea behind this gradient approximation was derived from meta-learning [Finn *et al.*, 2017]. **It is to adapt θ by using only a single training step**, rather than training until convergence completely solves the lower-level optimization (Eq. 11). The first term of the approximated upper gradient in Eq. 13 can be expanded according to the chain rule of derivation:

$$\begin{aligned}
&\nabla_\phi \mathcal{L}_{val}(\theta^{'}(\phi), \phi) \\
=&\nabla_\phi \theta^{'}(\phi) \cdot \nabla_{\theta^{'}} \mathcal{L}_{val}(\theta^{'}, \phi) + \nabla_\phi \phi \cdot \nabla_\phi \mathcal{L}_{val}(\theta^{'}, \phi) \\
=&-\alpha_\theta \nabla^2_{\phi,\theta} \mathcal{L}_{train}(\theta, \phi) \nabla_{\theta^{'}} \mathcal{L}_{val}(\theta^{'}, \phi) + \nabla_\phi \mathcal{L}_{val}(\theta^{'}, \phi),
\end{aligned} \tag{14}$$

where $\theta^{'} = \theta - \alpha_\theta \nabla_\theta \mathcal{L}_{train}(\theta, \phi)$ denotes the parameters for a one-step forward policy model. The matrix-vector product is calculated inefficiently in the expression above. Fortunately, we could reduce computational complexity by using Taylor's formula to approximate the product of the first-order and second-order gradients. Let ϵ be a small scalar, we got:

$$\begin{aligned}
&\nabla^2_{\phi,\theta} \mathcal{L}_{train}(\theta, \phi) \nabla_{\theta^{'}} \mathcal{L}_{val}(\theta^{'}, \phi) \\
\approx &\frac{1}{2\epsilon} \Big(\nabla_\phi \mathcal{L}_{train}(\theta + \epsilon \nabla_{\theta^{'}} \mathcal{L}_{val}(\theta^{'}, \phi), \phi) \\
&\qquad - \nabla_\phi \mathcal{L}_{train}(\theta - \epsilon \nabla_{\theta^{'}} \mathcal{L}_{val}(\theta^{'}, \phi), \phi) \Big).
\end{aligned} \tag{15}$$

In addition, the reward model details and training procedure are given in Appendix B.

4 Experiments

This section assesses the effectiveness of our approach across various offline RL benchmarks, highlighting the benefits of utilizing redistributed rewards in long-delayed settings. We aim to answer the following questions through our experiments: (1) Does the learned shaping reward function improve policy optimization in long-delayed reward environments? (2) How does the shaping reward function influence the performance of the DT model in the long-delayed setting? Additionally, we provide a case study that visualizes and interprets the learned shaping reward function.

4.1 Experimental Settings

We evaluated our method on both discrete and continuous control tasks. The discrete control tasks, including Atari [Bellemare *et al.*, 2015] and Minigrid [Chevalier-Boisvert *et al.*, 2018], involve high-dimensional observation spaces and require long-term reward redistribution. On the other hand, the continuous control tasks, such as OpenAI Gym Mujoco [Brockman *et al.*, 2016], Maze2d [Fu *et al.*, 2020], and FrankaKitchen [Fu *et al.*, 2020], not only have extremely delayed rewards but also require fine-grained continuous control. These benchmarks cover a wide range of tasks, spanning from easy to complex with varying time horizons. To adapt the offline data with dense rewards (e.g., Atari, Mujoco) to the episodic-reward setting, we made adjustments. We set the reward to zero in non-terminal states, moving away from per-step rewards. In the final step of the trajectory, we assigned the reward as the trajectory reward $R_{ep}(\tau)$, which was calculated by summing the instant rewards per step in the standard setting. For more detailed information on the experimental settings, please refer to Appendix C.

4.2 Baselines

We migrated five reward decomposition baselines including AVG, IRCR [Gangwani *et al.*, 2020], GAIL [Ho and Ermon, 2016; Guo *et al.*, 2018], RUDDER [Arjona-Medina *et al.*, 2019], and RRD [Efroni *et al.*, 2021; Ren *et al.*, 2022]. The details of baseline algorithms are given in Appendix C.2.

4.3 Overall Performance Comparison

Table 1 presents the performance evaluation of the DT model on various tasks, considering both dense and long-delayed reward settings. The tasks can be broadly classified into two categories. The first category comprises goal-oriented tasks, such as Minigrid and Maze2d, where the primary objective is to achieve a specific goal, like reaching the endpoint of a maze. In these tasks, instant environmental rewards are absent, and a binary value of 0 or 1 is provided

Benchmark	Dataset	DT with Dense Reward	DT with Sparse Reward	+AVG	+IRCR	+GAIL	+RUDDER	+RRD	+(Ours.)
Atari	Breakout	94.65	65.61	27.97	38.78	22.64	3.82	29.52	**71.91**
	Seaquest	1129.30	99.60	526.00	398.80	578.60	357.20	438.80	**688.3**
	Qbert	2215.80	224.00	513.50	830.00	770.75	2239.00	390.00	**3250.5**
	Pong	19.10	-20.98	-16.60	-20.28	-20.28	-20.43	-9.96	**11.27**
Minigrid	DoorKey-6x6	-	0.82	0.70	0.23	0.63	0.90	0.21	**0.91**
	DoorKey-8x8	-	0.79	0.84	0.11	0.56	0.82	**0.87**	**0.87**
	RedBlueDoor-6x6	-	0.81	0.82	0.41	**0.90**	0.83	0.89	**0.90**
	RedBlueDoor-8x8	-	0.50	0.49	0.33	0.53	0.52	0.34	**0.55**
	FourRooms	-	0.37	0.37	0.36	0.35	0.37	0.37	**0.38**
	DynamicObstacles-6x6	-	0.90	0.00	-0.97	0.90	0.90	0.90	**0.91**
D4RL	Maze2D-large	-	0.68	0.66	0.57	0.51	0.70	0.61	**0.75**
	FrankaKitchen	1.78	1.58	0.35	1.64	1.95	1.63	1.58	**2.07**
	Mujoco-halfcheetah	10131.42	10642.55	10437.06	9749.25	10349.96	10187.07	9914.28	**10651.21**
	Mujoco-walker2d	4965.35	4964.44	4836.90	4906.37	4913.68	4856.62	4965.56	**4970.34**
	Mujoco-hopper	3344.68	2896.20	2536.87	3032.79	3183.56	3327.37	3073.67	**3336.45**
	Mujoco-ant	4475.76	4554.55	4592.66	4219.80	4629.40	3626.17	4644.63	**4711.73**
Count.				0	0	1	0	1	**16**

Table 1: Average accumulated rewards on tasks from three benchmarks are presented using dense and sparse reward settings. The results are based on 100 runs, and the best mean scores are highlighted in bold, while the second-best mean scores are underlined. Tasks denoted by "-" are goal-oriented, with only trajectory rewards of 0 or 1, making them unsuitable for training with dense environmental rewards.

at the last timestep, indicating the trajectory reward for the entire trajectory and signifying failure or success. The second category consists of score-oriented tasks like Atari and Mujoco, where obtaining higher scores requires precise continuous controls. For instance, in Atari's Pong game, the aim is to win as many rounds as possible. We first evaluated the DT model's performance on score-oriented tasks under both dense rewards (using instant rewards) and sparse rewards (using trajectory rewards). The absence of instant reward feedback significantly and visibly deteriorated the performance of the DT model. This decline is evident from the noticeable increase in the fitting error of the model curve, as shown in Figure 5 in Appendix C, and the substantial decrease in the accumulated reward, as reported in Table 1. These findings substantiate our hypothesis that the DT model encounters difficulties in capturing complex dependencies related to rewards in environments characterized by long delays. Additionally, we solely reported performance results with sparse rewards on goal-oriented tasks, as these tasks inherently involve episodic-reward settings without any dense rewards.

We assessed the performance of six offline reward redistribution models, including ours, on three benchmarks that utilize episodic-reward settings. The aim was to compare the effectiveness of these approaches in enhancing the performance of the DT model. By enabling the search for the globally optimal solution of the reward function, our DTRD facilitated the learning of improved policies from offline data across nearly all episodic-reward setting tasks. As shown in Table 1, DTRD demonstrated a significant performance improvement on the Atari benchmark, achieving an average improvement of over 5 times compared to the original DT in long-delayed settings. Similarly, on the Minigrid and D4RL benchmarks, DTRD achieved an average performance improvement of approximately 10% and 8% respectively. Furthermore, although most of the evaluated methods effectively improve the learning performance of the DT model in environments with long-delayed rewards by redistributing rewards to shape rtgs, DTRD consistently outperformed baseline algorithms in all evaluated tasks. In contrast, baseline methods such as IRCR, GASIL, RUDDER, and RRD were policy-independent, relying on heuristic design or parameterization approximation to learn a reward function from long-delayed trajectories. However, the experimental results in Table 1 reveal that these policy-independent reward modeling methods often become trapped in local optima. This limitation impedes the optimization of the policy model and amplifies the negative impact of suboptimal reward information on policy learning. The fragmented nature of reward function learning in these methods contributes to this issue, as their optimization objectives solely focus on aligning the cumulative reward sum for redistribution with the trajectory reward of each trajectory, without considering feedback on the quality of policy learning. Thus, it is crucial for the optimization objective to directly assess whether redistributed rewards can genuinely enhance the performance of the DT model. This optimization objective is precisely what DTRD considered, which contributes significantly to outperforming baseline models.

In the series of tasks, such as Atari and Minigrid for discrete actions and Maze and Kitchen for continuous actions, DT models are more negatively affected by the long-delayed setting. To tackle this issue, our approach introduced a joint learning method combining the reward and policy models through bi-level optimization. As shown in Table 1, DTRD notably improved the learning and generalization ability of DT models across these tasks by serving the validation performance of the policy model as an explicit objective to guide the learning of the reward function. Figure 5 in Appendix C displays the convergence curves of DTRD on the Atari benchmark. DTRD exhibited substantially lower training and validation errors than the DT model in the long-delayed setting. Furthermore, the convergence error of DTRD closely resem-

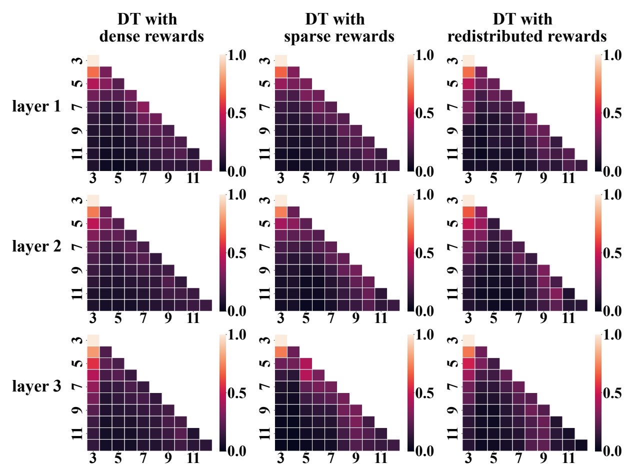


Figure 3: The attention weights of DT models in the Pong environment under dense, sparse, and redistributed reward settings. The horizontal axis represents the input timestep, while the vertical axis represents the output timestep. Brighter colors in the boxes indicate higher attention weights to the corresponding positions.

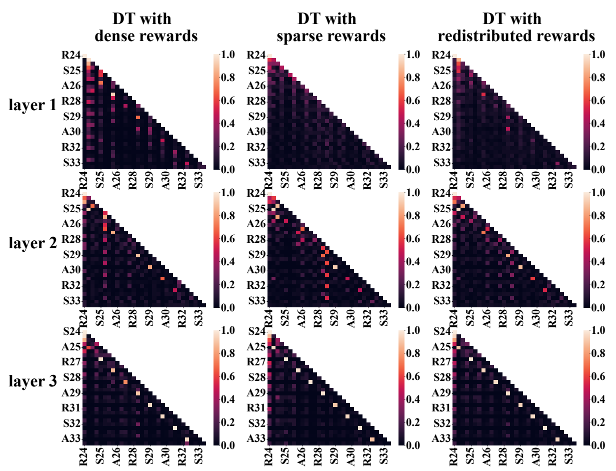


Figure 4: Attention weights of DT models in the halfcheetah environment under three reward settings. The horizontal and vertical axes represent rtgs, states, and actions at different timesteps in the input and output. Brighter colors indicate higher attention weights. The distribution of attention matrices is similar across all cases, showing a bright line tilted 45 degrees to the lower right.

bled that of the DT model in the dense-reward setting. These results indicated that our approach's redistributed rewards significantly enhance the DT model's training and convergence. Notably, in the Mujoco benchmark where the original rewards were already slightly delayed, DTRD also demonstrated a slight performance improvement, it even worked better than the dense reward setting with original rewards. Additionally, DTRD outperformed all baseline methods in both performance and stability. The baseline methods demonstrated considerable variability in performance across different environments, whereas DTRD consistently performed well across the majority of experimental environments. Further analysis such as experiments on delayed length and hyper-parameter sensitivity can be found in Appendix C.

4.4 Case Study

In Section 4.3, we conducted experiments to demonstrate the significant performance degradation of DT models in the sparse-reward setting, particularly in the Atari benchmark. Conversely, we observed that rewards have a lesser impact on DT model performance in the Mujoco benchmark. In this section, we analyze the effect of rewards on the learning of DT models by examining attention-weight matrix images in several selected cases across different environments.

The Pong environment from Atari. We intercepted a 40-step trajectory as a context sequence in the Pong environment and inputted it into the DT model with different reward settings to observe the attention matrix. Since the original attention matrix is high-dimensional and challenging to visualize, we applied average pooling to reduce its dimensionality for presentation. Figure 3 shows that sparse rewards pose difficulties in learning the DT model, resulting in biased attention dependencies and varying dependencies. However, the redistributed rewards effectively correct these biases. In the dense-reward setting, the DT model's attention weight at timestep t is mainly influenced by elements like rtgs at the previous timestep $t-1$ and the first timestep of the context sequence. When instant rewards are delayed until the final state, the inputs on most layers of the DT model lack dependencies on the first rtg of the entire context sequence. Consequently, the dependencies from the first rtg are almost entirely disregarded during action generation in several later steps of the context sequence. Fortunately, DTRD successfully addressed this bias in the attention matrix. With the aid of redistributed rewards, the DT model reinstated the dependencies on the first rtg and closely approximated the attention matrix of the dense-reward setting.

The Halfcheetah environment from Mujoco. We analyzed the attention distribution of rtgs (R), states (S), and actions (A) in the halfcheetah environment by intercepting a 10-step trajectory as the context sequence. In both two reward settings, we observed that the DT model's decision a_t at timestep t heavily relied on dependencies with the states s_t and s_{t-1}, while showing minimal dependencies on historical actions, and rtgs, as shown in Figure 4.

5 Conclusion

In this paper, we proposed DTRD , an adaptive reward redistribution algorithm that addressed the issue of model degradation in DT models due to the accumulation of unwanted biases in autoregressive learning global signals due to reward delays. DTRD incorporated the validation error of the policy model to guide the reward modeling optimization process, enabling simultaneous learning of the reward model and policy model using a bi-level optimization algorithm. Experimental results demonstrated that DTRD effectively mitigates performance degradation in DT models in long-delayed settings, surpassing other existing reward redistribution methods. Furthermore, DTRD holds potential for generalization to other transformer-based offline RL models, such as TT [Janner *et al.*, 2021]. We leave these investigations as our future work.

Acknowledgements

This work was supported by the National Key R&D Program of China (2021ZD0113903), grants from the Natural Science Foundation of China (62225202, 62202029), and the Outstanding Research Project of Shen Yuan Honors College, BUAA, (No.230121208). Thanks for the computing infrastructure provided by Beijing Advanced Innovation Center for Big Data and Brain Computing. This work was also sponsored by CAAI-Huawei MindSpore Open Fund.

References

[Arjona-Medina *et al.*, 2019] Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. RUDDER: return decomposition for delayed rewards. In *NeurIPS*, pages 13544–13555, 2019.

[Bellemare *et al.*, 2015] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents (extended abstract). In *IJCAI*, pages 4148–4152, 2015.

[Bellman, 1966] Richard E. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Chen *et al.*, 2021] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *NeurIPS*, pages 15084–15097, 2021.

[Chevalier-Boisvert *et al.*, 2018] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018. Accessed: 2023-06-25.

[Chung *et al.*, 2014] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NeurIPS*, 2014.

[Devlin *et al.*, 2011] Sam Devlin, Daniel Kudenko, and Marek Grzes. An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Adv. Complex Syst.*, 14(2):251–278, 2011.

[Devlin *et al.*, 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186, 2019.

[Dosovitskiy *et al.*, 2021] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[Efroni *et al.*, 2021] Yonathan Efroni, Nadav Merlis, and Shie Mannor. Reinforcement learning with trajectory feedback. In *AAAI*, 2021.

[Emmons *et al.*, 2022] Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. Rvs: What is essential for offline RL via supervised learning? In *ICLR*, 2022.

[Finn *et al.*, 2017] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, volume 70, pages 1126–1135, 2017.

[Fu *et al.*, 2018] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *ICLR*, 2018.

[Fu *et al.*, 2020] 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.

[Fu *et al.*, 2021] Xingcheng Fu, Jianxin Li, Jia Wu, Qingyun Sun, Cheng Ji, Senzhang Wang, Jiajun Tan, Hao Peng, and S Yu Philip. Ace-hgnn: Adaptive curvature exploration hyperbolic graph neural network. In *ICDM*, pages 111–120, 2021.

[Gangwani *et al.*, 2019] Tanmay Gangwani, Qiang Liu, and Jian Peng. Learning self-imitating diverse policies. In *ICLR*, 2019.

[Gangwani *et al.*, 2020] Tanmay Gangwani, Yuan Zhou, and Jian Peng. Learning guidance rewards with trajectory-space smoothing. In *NeurIPS*, 2020.

[Guo *et al.*, 2016] Xiaoxiao Guo, Satinder Singh, Richard L. Lewis, and Honglak Lee. Deep learning for reward design to improve monte carlo tree search in ATARI games. In *IJCAI*, pages 1519–1525, 2016.

[Guo *et al.*, 2018] Yijie Guo, Junhyuk Oh, Satinder Singh, and Honglak Lee. Generative adversarial self-imitation learning. *arXiv preprint arXiv:1812.00950*, 2018.

[Harutyunyan *et al.*, 2019] Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Gheshlaghi Azar, Bilal Piot, Nicolas Heess, Hado van Hasselt, Gregory Wayne, Satinder Singh, Doina Precup, and Rémi Munos. Hindsight credit assignment. In *NeurIPS*, pages 12467–12476, 2019.

[Ho and Ermon, 2016] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *NeurIPS*, pages 4565–4573, 2016.

[Hu *et al.*, 2020] Yujing Hu, Weixun Wang, Hangtian Jia, Yixiang Wang, Yingfeng Chen, Jianye Hao, Feng Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping. In *NeurIPS*, 2020.

[Janner *et al.*, 2021] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *NeurIPS*, pages 1273–1286, 2021.

[Jiang *et al.*, 2016] Nan Jiang, Alex Kulesza, Satinder Singh, and Richard L. Lewis. The dependence of effective planning horizon on model accuracy. In *IJCAI*, pages 4180–4189, 2016.

[Li *et al.*, 2023] Jianxin Li, Qingyun Sun, Hao Peng, Beining Yang, Jia Wu, and S Yu Phillp. Adaptive subgraph neural network with reinforced critical structure mining. *TPAMI*, 2023.

[Lillicrap *et al.*, 2016] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[Liu *et al.*, 2018] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2018.

[Liu *et al.*, 2019] Yang Liu, Yunan Luo, Yuanyi Zhong, Xi Chen, Qiang Liu, and Jian Peng. Sequence modeling of temporal credit assignment for episodic reinforcement learning. *arXiv preprint arXiv:1905.13420*, 2019.

[Mataric, 1994] Maja J. Mataric. Reward functions for accelerated learning. In *ICML*, pages 181–189, 1994.

[Ng and Russell, 2000] Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *ICML*, pages 663–670, 2000.

[Ng *et al.*, 1999] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, pages 278–287, 1999.

[Petrik and Scherrer, 2008] Marek Petrik and Bruno Scherrer. Biasing approximate dynamic programming with a lower discount factor. In *NeurIPS*, pages 1265–1272, 2008.

[Radford and Narasimhan, 2018] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. https://openai.com/research/language-unsupervised, 2018. Accessed: 2023-06-25.

[Rajeswaran *et al.*, 2020] Aravind Rajeswaran, Igor Mordatch, and Vikash Kumar. A game theoretic framework for model based reinforcement learning. In *ICML*, volume 119, pages 7953–7963, 2020.

[Ren *et al.*, 2022] Zhizhou Ren, Ruihan Guo, Yuan Zhou, and Jian Peng. Learning long-term reward redistribution via randomized return decomposition. *ICLR*, 2022.

[Schmidhuber, 2019] Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards - just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

[Song *et al.*, 2019] Shihong Song, Jiayi Weng, Hang Su, Dong Yan, Haosheng Zou, and Jun Zhu. Playing FPS games with environment-aware hierarchical reinforcement learning. In *IJCAI*, pages 3475–3482, 2019.

[Sorg *et al.*, 2010] Jonathan Sorg, Satinder Singh, and Richard L. Lewis. Reward design via online gradient ascent. In *NeurIPS*, pages 2190–2198, 2010.

[Srivastava *et al.*, 2019] Rupesh Kumar Srivastava, Pranav Shyam, Filipe Wall Mutz, Wojciech Jaśkowski, and Jürgen Schmidhuber. Training agents using upside-down reinforcement learning. *arXiv preprint arXiv:1912.02877*, 2019.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. *IEEE Trans. Neural Networks*, 9(5):1054–1054, 1998.

[Sutton, 1988] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Mach. Learn.*, 3:9–44, 1988.

[Torabi *et al.*, 2018] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In *IJCAI*, page 4950–4957, 2018.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[Wu and Tian, 2017] Yuxin Wu and Yuandong Tian. Training agent for first-person shooter game with actor-critic curriculum learning. In *ICLR*, 2017.

[Xu *et al.*, 2018] Zhongwen Xu, Hado van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *NeurIPS*, pages 2402–2413, 2018.

[Yang *et al.*, 2023] Yang Yang, Yurui Huang, Weili Guo, Baohua Xu, and Dingyin Xia. Towards global video scene segmentation with context-aware transformer. In *AAAI*, 2023.

[Zheng *et al.*, 2018] Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. In *NeurIPS*, pages 4649–4659, 2018.

[Zheng *et al.*, 2020] Zeyu Zheng, Junhyuk Oh, Matteo Hessel, Zhongwen Xu, Manuel Kroiss, Hado van Hasselt, David Silver, and Satinder Singh. What can learned intrinsic rewards capture? In *ICML*, volume 119, pages 11436–11446, 2020.

[Zheng *et al.*, 2022] Zeyu Zheng, Risto Vuorio, Richard L. Lewis, and Satinder Singh. Adaptive pairwise weights for temporal credit assignment. In *AAAI*, pages 9225–9232, 2022.

[Zhou *et al.*, 2021] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *AAAI*, pages 11106–11115, 2021.