algorithms

MDPI

Article

Optimizing Reinforcement Learning Using a Generative Action-Translator Transformer

Jiaming Li [1], Ning Xie [1,*] and Tingting Zhao [2,3]

[1] Center for Future Media, School of Computer Science and Engineering, and Yibin Park, University of Electronic Science and Technology of China, Chengdu 611731, China; 202122080124@std.uestc.edu.cn
[2] College of Artificial Intelligence, Tianjin University of Science and Technology, Tianjin 300457, China; tingting@tust.edu.cn
[3] RIKEN Center for Advanced Intelligence Project (AIP), Tokyo 103-0027, Japan
* Correspondence: xiening@uestc.edu.cn or seanxiening@gmail.com; Tel.: +86-1779-640-6627

Abstract: In recent years, with the rapid advancements in Natural Language Processing (NLP) technologies, large models have become widespread. Traditional reinforcement learning algorithms have also started experimenting with language models to optimize training. However, they still fundamentally rely on the Markov Decision Process (MDP) for reinforcement learning, and do not fully exploit the advantages of language models for dealing with long sequences of problems. The Decision Transformer (DT) introduced in 2021 is the initial effort to completely transform the reinforcement learning problem into a challenge within the NLP domain. It attempts to use text generation techniques to create reinforcement learning trajectories, addressing the issue of finding optimal trajectories. However, the article places the training trajectory data of reinforcement learning directly into a basic language model for training. Its aim is to predict the entire trajectory, encompassing state and reward information. This approach deviates from the reinforcement learning training objective of finding the optimal action. Furthermore, it generates redundant information in the output, impacting the final training effectiveness of the agent. This paper proposes a more reasonable network model structure, the Action-Translator Transformer (ATT), to predict only the next action of the agent. This makes the language model more interpretable for the reinforcement learning problem. We test our model in simulated gaming scenarios and compare it with current mainstream methods in the offline reinforcement learning field. Based on the presented experimental results, our model demonstrates superior performance. We hope that introducing this model will inspire new ideas and solutions for combining language models and reinforcement learning, providing fresh perspectives for offline reinforcement learning research.

Keywords: machine learning; reinforcement learning; Transformer; action prediction; Action-Translator Transformer

check for updates

Citation: Li, J.; Xie, N.; Zhao, T. Optimizing Reinforcement Learning Using a Generative Action-Translator Transformer. *Algorithms* **2024**, *17*, 37. https://doi.org/10.3390/a17010037

Academic Editor: Frank Werner

Received: 14 December 2023
Revised: 8 January 2024
Accepted: 11 January 2024
Published: 16 January 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In recent years, with the continuous development and progress in the field of artificial intelligence, AI has achieved remarkable success in many areas, even surpassing human performance in scenarios such as board games and electronic games [1]. In the field of reinforcement learning, offline reinforcement learning is a crucial research area. Unlike online reinforcement learning, which requires the training process to occur in a real environment or a simulated one, offline reinforcement learning relies solely on historical offline datasets. This allows the training of policy or value functions for complex environments that are either difficult to model or entail high execution costs without the need for environment simulation. However, current algorithms based on offline reinforcement learning are mostly built on the theoretical foundation of traditional MDP and suffer from the problem of distribution shift [2], where the training policy differs from the behavior policy.

With the continuous development in the field of NLP from BERT [3] to GPT-3 [4] and the latest ChatGPT [5], interdisciplinary integration has become increasingly common. NLP models and algorithms are being experimented with in the field of reinforcement learning. The classic NLP model Transformer has been integrated into reinforcement learning algorithms, as the nature of reinforcement learning itself involves causal sequences that require the transmission of historical information for training the next step. Although early models like Long Short-Term Memory (LSTM) [6] and Gate Recurrent Unit (GRU) [7] have been used in reinforcement learning, the Transformer [8], with its powerful memory function, is now being embedded into reinforcement learning. However, due to the peculiarities of Transformer model training, it requires substantial data and computational resources. Current research on the combination of Transformer and reinforcement learning primarily focuses on integrating the Transformer into the training process of traditional reinforcement learning algorithms and addressing the issue of mismatch. Based on this, the proposed algorithms include GTrXL [9] and CoBERL [10]. However, this simple integration has not only failed to address the existing problems in traditional reinforcement learning, but has also introduced new issues, such as unstable Transformer training and the need for extensive datasets.

With the introduction of the Decision Transformer (DT) [11] algorithm, a new domain has been opened up in the training of reinforcement learning. The underlying framework of the original reinforcement learning training, namely the Markov decision process, is abandoned. Instead, the language model is directly employed as the training framework for reinforcement learning, and the training process is conducted based on the GPT model. This aims to find the optimal trajectory and solve the reinforcement learning problem. Since language models often need to be built on large data sets for training [12], this model method can only be applied to the field of offline reinforcement learning. Because of its simple structure and being suitable for text generation tasks, the GPT model has excellent performance in the optimal trajectory generation of reinforcement learning. However, the model also has unreasonable aspects, leading to a lack of interpretability in the model. During the training process of DT, the entire offline trajectory is used directly as the training data input for GPT, serving as historical information to predict the optimal action at the next time step. However, when the DT model outputs this optimal action, it also includes the next time step's state and reward information in the output. State and reward information change automatically during the reinforcement learning training process in conjunction with the action, and should not be considered as model outputs to calculate the loss function. Additionally, these extra outputs increase the training difficulty of the model because the performance of a language model itself depends on the length of the input sequence. Introducing unnecessary input–output pairs will reduce the training effectiveness of the model. We will provide a detailed introduction to DT in the related work and validate its shortcomings.

In order to explore more possibilities for combining language models with reinforcement learning and enhance the adaptability of the model to the reinforcement learning training process, this paper proposes a novel intelligent agent training model based on a text translation model in the context of training game AI in offline reinforcement learning. The main contributions of this paper are as follows:

- A novel sequence-decision-based reinforcement learning method: We introduce the ATT network model, which is built upon the Sequence to Sequence model structure used in text translation tasks. It predicts actions based on observable scene information, utilizes the Transformer model to identify trajectories with the maximum reward return in different game environments, and is the first to employ the translation task framework to align with the reinforcement learning task.
- Encoding form adapted for text translation models: We devise a unique encoding form based on the original elements of reinforcement learning (state, action, reward) and introduce positional information to tailor it for the language model's training process.

- Based on a review of the existing literature, we analyze future development directions and challenges from the perspectives of reinforcement learning algorithms and other tasks in the field of natural language processing. The purpose of this discussion is to help researchers better comprehend the key aspects of combining reinforcement learning with large models and encourage the application of more language models in reinforcement learning tasks.

The structure of the subsequent sections of this paper is as follows: In Section 2, we will introduce the relevant background knowledge and theoretical system of reinforcement learning and natural language processing. We will briefly highlight algorithms that perform well in the combined field. In Section 3, our focus will be on introducing the ATT model, covering the problems it addresses, details of the method design, model architecture, and the training and inference processes. In Section 4, we will conduct specific verifications of ATT experiments, including an introduction to the experimental environment and an analysis of the experimental results. In Section 5, based on a summary of existing methods and the proposed method in this paper, we will discuss future research directions and challenges of combining reinforcement learning with NLP.

2. Background

From the initial proposal of the Markov theoretical system, based on Bellman's optimal equation, various methods have been suggested to tackle the reinforcement learning problem. With the advent of deep learning, Convolutional Neural Networks (CNN) found application across diverse fields. Simultaneously, reinforcement learning capitalized on the neural network's capacity for processing high-dimensional problems. It mapped some of the original complex multi-dimensional problems onto neural networks, leading to the development of highly efficient deep reinforcement learning algorithms. In recent years, NLP models have experienced rapid development, prompting an increasing number of researchers to explore the potential combination of language modeling with reinforcement learning. In this section, we will briefly introduce the relevant theoretical foundations of reinforcement learning and natural language processing. Concurrently, we will outline various attempts made by researchers to integrate the two fields. Finally, we will introduce the Decision Transformer, a highly successful model that combines NLP and RL. This model will serve as a key point of comparison, and we will explain its relevant structure and theory to facilitate understanding of the subsequent improvements made in this paper.

2.1. Markov Decision Process (MDP)

Reinforcement learning centers around the sequential decision-making process in the interaction between an agent (as the subject) and the environment (as the object). In a specific environment, there exists an agent capable of perceiving the current environmental information and generating actions based on this information. These actions lead to changes in the observed environmental information for the agent and, simultaneously, a reward signal is provided to the agent when the environmental state undergoes a change. This process continues until the agent ceases its actions or the environment reaches a terminal state. The objective of reinforcement learning is to train the agent's action policy, enabling it to attain the maximum cumulative reward during interaction with the environment [13]. MDP is a classical model for intelligent decision-making. Its core theory posits that if the future state of a state is independent of its past state, i.e., it depends solely on the present state, then this state possesses the Markov property. We define the information involved in this process as follows:

- S: a finite set of states, where s_i represents the state at step i.
- A: a finite set of actions, where a_i represents the action at step i.
- $P_{s,a}$: the probability of state transition, which denotes the probability of transitioning to other states given the action a in the current state s.
- R: the immediate or expected reward obtained from the state transition.
- γ: the discount factor that adjusts the influence of future rewards on the current state.

Thus, MDP can be represented as a quintuple:

$$M = (S, A, P_{s,a}, R, \gamma) \tag{1}$$

Due to the existence of the reward value R, the action a chosen by the agent is no longer random, but is aimed at obtaining the maximum cumulative reward. The state transition in this system is determined by both the current state s_t and the action taken a_t, the next state s_{t+1} is determined by the transition probability model $P(s_{t+1}|s_t, a_t)$, and the reward value r_t obtained at the current time is also determined by the probability distribution $P(r_t|s_t, a_t)$.

2.2. Reinforcement Learning Algorithms

Currently, mainstream reinforcement learning algorithms train agents based on modeling the environment with MDP. When agents need to make actions interacting with the environment, they either follow the current policy or use the current value function. In deep reinforcement learning, policies and value functions are often parameterized by variables in deep neural networks. Subsequently, gradient-based methods are employed for optimization. Loss is calculated using a loss function, followed by backpropagation to update network parameters until the value or policy function meets the training objectives. Based on the mentioned training approaches, mainstream reinforcement learning methods can be categorized into three major classes: Value-Based, Policy-Based, and Actor-Critic algorithms that simultaneously rely on both policy and value functions.

Value-Based: This method requires us to define a value function. For tasks with a small state space and action space, we can use a tabular form to represent the value function, such as Q-Learning. It optimizes a state-action value table where each data entry represents the immediate reward obtained by taking action a in state s, denoted as $Q(s, a)$. The core update formula is as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2}$$

Here, α represents the update step size, and the value function $Q(s_t, a_t)$ is continuously updated through iterations until convergence. Based on the action-value function, the optimal action in a certain state can be selected. In high-dimensional and complex environments, using a table alone is insufficient to represent all states and actions. Therefore, we use neural networks to fit the value function, leading to the birth of the classic deep reinforcement learning algorithm, Deep Q-Network (DQN) [14], which uses parameters θ to approximate the value function. Subsequent Value-Based algorithms mainly focus on further optimizing the direction of reducing errors and improving sample learning efficiency.

Policy-Based: Unlike value-based methods, policy-based methods directly learn parameterized policies π_θ. This allows us to avoid the need to solve the value function for each action in the space. Therefore, this method is suitable for solving problems with high-dimensional or continuous action spaces. It intuitively improves the performance of the policy π_θ in the parameter space through gradient ascent methods and policy iteration updates, achieving the maximization of cumulative rewards. For example, the Policy Gradient (PG) [15] algorithm has the objective function to maximize the expected return by adjusting θ, represented as follows:

$$J(\pi_\theta) = E_{\pi \sim \tau}[R(\tau)] \tag{3}$$

In the above equation, τ represents a complete trajectory from the beginning to the end, π represents the current policy of the agent, and $\pi \sim \tau$ represents the action trajectory of the agent sampled under this policy. For this maximization problem, we use the gradient

ascent algorithm to find its maximum value, and the optimization formula for parameter θ is as follows:

$$\theta^* = \theta + \alpha \nabla J(\pi_\theta) \tag{4}$$

The essence of policy-based algorithms lies in determining the gradient of the final return function $J(\pi_\theta)$ with respect to θ, commonly referred to as the policy gradient. This forms the basis for the optimization of various mainstream policy-based algorithms like PPO, TRPO, etc.

Actor-Critic: This approach, which integrates both policy and value, concurrently learns an actor function and a critic function. The actor function serves as the agent's policy function $\pi(\cdot|s)$, determining the action the agent will take in the current state. Meanwhile, the critic function functions as the state value function, providing an assessment of the action taken by the agent [16]. In this context, the Actor network is implemented using the policy gradient algorithm. For the Critic network, the Actor-Critic algorithm adopts the bootstrap method for estimating the Q-value function, and it learns the Critic function by minimizing the temporal-difference error, with the loss function given by:

$$J_Q = (r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))^2 \tag{5}$$

where the action a_{t+1} is obtained by sampling the current policy π_θ in state s_{t+1}. Algorithms arising from the Actor-Critic base such as A3C are also important algorithms in reinforcement learning.

2.3. Transformer

The Transformer was introduced by the Google Brain team in 2017 to address the limitation of parallelizing computations using RNN in NLP. RNN-based models faced inefficiency issues due to their inability to parallelize, making them less effective. The Transformer is a deep learning model built entirely on the self-attention mechanism. In NLP tasks, when dealing with text input data after Tokenization and Embedding processing, RNN structures would require feeding each token individually into the model. This is because RNNs rely on hidden state information passed from the previous input for each subsequent input. However, the attention mechanism in the Transformer allows the entire sequence to be input into the model. It ensures the sequence's order by introducing positional coding information and utilizes masking to facilitate parallel training. The Transformer's increased complexity results in higher accuracy and performance compared to RNN. The basic structure is illustrated in Figure 1:

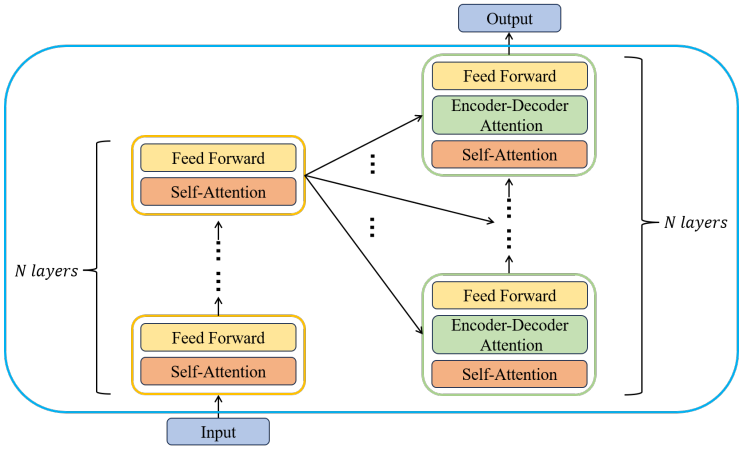

Figure 1. The overall framework of the Transformer model consists mainly of Encoders and Decoders. It is composed of six small Encoder and Decoder structures. The Encoders on the left process contextual text and calculate correlations between texts using the attention mechanism. On the right, the Decoders handle textual correlations for text output. The output form varies depending on the type of task.

Taking the text translation task as an example, during the pre-training stage, the input text data undergoes a series of steps. Firstly, the text is tokenized, breaking it into a sequence of words. Subsequently, the words are vectorized through embedding, where each word is input, and a corresponding vector is output. Before feeding these embedded vectors into the Transformer model, they are further encoded with position-related information. This positional encoding can be manually computed by presetting the positional encoding information. Following this, self-attention is computed for each position, expressing the correlation between different positions by calculating attention values for each position relative to other positions. The underlying principle is that when attention is given to data in the current position, information from other positions is also considered. The attention mechanism regulates the extent to which attention is given to other positions. The formula for calculating self-attention is as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{6}$$

where Q (Query), K (Key), and V (Value) are obtained by multiplying the encoded matrix by a weight matrix, and d_k is the dimensionality of our data after encoding. The self-attention mechanism allows positions to be interconnected, addressing the challenge of forgetfulness associated with longer sequences in the original LSTM and GRU models.

Another significant factor enabling parallel training in the Transformer is the introduction of the Mask mechanism. In the Decoder side, we input the translated text as a whole, but the Mask restricts the visibility of future words when predicting the word at a specific position. This approach ensures that the model leverages information from preceding words to make predictions for the current position, enhancing its ability to utilize previous context effectively during parallel training. By preventing the model from accessing actual future data during parallel training, this method maintains training effectiveness.

2.4. Transformer in RL

Reinforcement learning provides a mathematical framework for solving sequential decision problems and determining optimal actions through formulas. A persistent challenge in reinforcement learning is sample utilization. Inadequate sample utilization necessitates more training data and increased agent-environment interactions, significantly raising algorithm training overhead. This limitation hampers the agent's ability to learn an effective policy. Researchers have proposed various solutions to address this issue, including the utilization of buffers, autonomous construction of environment models, and more. However, most RL architectures are built upon methods related to supervised learning and semi-supervised learning. In high-dimensional scenarios, CNN serves as a function approximator, while for partially observable environments or situations requiring information retention, RNN processes the feature information.

The Transformer model has garnered considerable attention since its introduction, demonstrating superior performance compared to CNN and RNN. It exhibits excellent capabilities in handling long sequence problems, possesses strong scalability, and has evolved into a prevalent paradigm for numerous supervised learning tasks. Consequently, researchers have initiated efforts to apply the Transformer architecture to the realm of reinforcement learning.

Analogous to the use of neural networks in RL, researchers initially attempted to employ the Transformer as a function approximator in RL algorithms. However, merely substituting Transformer for LSTM in reinforcement learning yielded poorly trained models. Mishra et al. [17] explored the application of Transformer to various simple RL tasks (e.g., Bandit task, tabular version of Markov process) and found that the performance only matched that of the as-you-go policy. The main reasons for this were analyzed as follows:

- The training parameters of Transformer are complex and demand substantial data and computational resources to converge, while RL itself suffers from low sample utilization.

- RL receives state observation information sequentially in chronological order, and the order is not explicitly given but must be learned by Transformer.
- RL algorithms are highly sensitive to the architecture of deep neural networks.

These challenges hinder the effectiveness of Transformer in the original framework of reinforcement learning. In 2019, Parisotto et al. [9] proposed the GTrXL framework, leveraging the capability of Transformer-XL [18] to learn more than a fixed-length dependency without disrupting temporal coherence. The modified structure of GTrXL demonstrated significant improvement over LSTM in environments requiring long-term memory, effectively addressing various long and short-term memory requirements. Simultaneously, this new network structure can be combined with various strategies, rendering Transformer more suitable for the optimization process of reinforcement learning. The GTrXL framework established a baseline for using Transformer in reinforcement learning, prompting the emergence of subsequent approaches that build upon it. For instance, Adaptive Transformer in RL [19] enhances the efficiency of the original framework by increasing the memory size of GTrXL and employing Adaptive Attention Span. Ultimately, it maintains similar performance to the original GTrXL with a notable increase in training speed and a reduction in resource consumption. CoBERL [10], inspired by the self-supervised training task of mask prediction in BERT, combines GTrXL and LSTM using Trainable Gate. It integrates the two-way mask prediction self-supervised task and BERT's comparison learning method, resulting in improved learning outcomes for this combined model.

All the aforementioned approaches operate within the original RL framework, aiming to investigate the adaptability of Transformer in RL and optimize the training setup to mitigate the impact of Transformer's inherent challenges on the training task. Essentially, they continue to address the reinforcement learning problem by solving the value function or the policy network. In the following, we will introduce a novel concept that maximizes the advantages of Transformer in solving sequence problems. This approach discards the traditional RL framework and has shown remarkable results.

2.5. Decision Transformer (DT)

The DT model was introduced in 2021 by Lili [11] and collaborators, pioneering a novel approach to integrating NLP models with reinforcement learning. Previously, the utilization of the Transformer in reinforcement learning primarily involved substituting a portion of the structure within the reinforcement learning algorithm, while the overall training framework still adhered to the principles of MDP. The DT algorithm distinguishes itself from traditional reinforcement learning in that it employs Transformer to directly learn the actions that the intelligent agent should take to achieve the desired reward, rather than focusing on learning strategies or value functions.

To align with the Transformer structure, DT processes the training data differently. For the traditional trajectory $(s_0, a_0, r_0, s_1, a_1, r_1, s_2, \ldots, s_t, a_t, r_t, s_{t+1})$, it needs to be transformed into the following format:

$$\tau = (\hat{R}_0, s_0, a_0, \hat{R}_1, s_1, a_1, \ldots, \hat{R}_t, s_t, a_t) \tag{7}$$

Here, a and s represent actions and states at a given moment, as in the original definition. However, $\hat{R}$ is not an immediate reward; instead, it represents the cumulative reward that can be earned from the current state to the end of the trajectory (Returns-to-go). This definition enables the conversion of reinforcement learning into a supervised learning form for training. The training framework is illustrated in Figure 2.

We input the encoded sequence data as a token (the row of circles beneath the Embedding layer in Figure 2) into our Transformer. Here, three elements of the same moment $(\hat{R}_t, s_t, a_t)$ share the same positional information since these data come from the same moment. We model the joint distribution of the total return, state, and action trajectories for subsequent trajectories through Transformer. During training, we use a mask to hide the subsequent information of the current moment. This ensures that the Transformer

can only use historical trajectories to infer the current action to be executed and output it. The optimization goal is the difference between predicted actions and real actions, using cross-entropy if the action is discrete and mean square error (MSE) if the action is continuous.

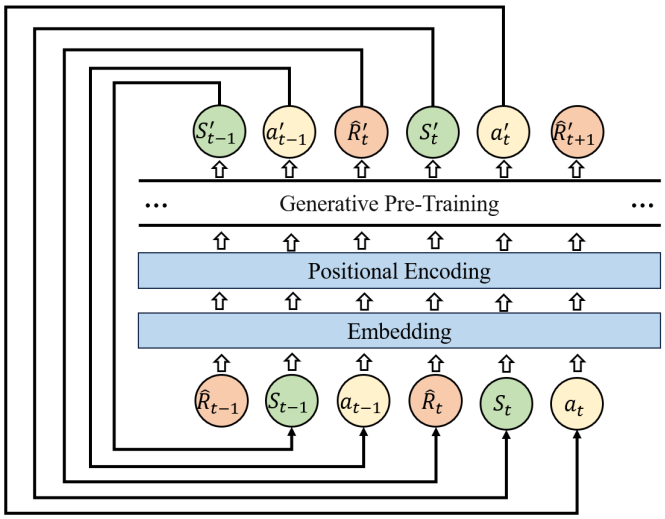

Figure 2. DT structure, which models reinforcement learning using a generative task approach, based on a sequence of inputs $(\hat{R}, s, a)$, with the entire sequence output intact at the output.

After training the network, we input the maximum reward value of the environment as our target reward (a priori knowledge). We then input the network and the current state, performing the derivation to obtain a trajectory that can yield the target reward. The outlined process represents the fundamental flow of the DT structure.

3. Action-Translator Transformer

3.1. Problem Description

The DT algorithm introduces a novel approach to solving reinforcement learning problems using NLP methods. Since the language model utilized demands extensive datasets for training, in online reinforcement learning, the data collected by the sample pool is often insufficient to support the training of the language model. Therefore, this method is primarily applied to offline reinforcement learning scenarios, leveraging a large repository of offline sample data to facilitate model training. In the DT algorithm, the model employed for training follows a GPT structure—a simplified version compared to the original Transformer, consisting of multiple Decoders stacked together. The structure is illustrated in Figure 3.

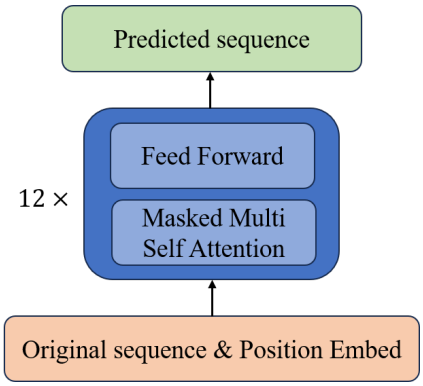

Figure 3. The language model (GPT) used in DT, which contains only one part with respect to Transformer, for the input of a text, predicts that text sequence in its entirety and is suitable for text generation tasks.

The GPT language model adopts a unidirectional structure without a clear distinction between Encoder and Decoder. To accommodate the input into the model, DT processes offline trajectory data into a single trajectory, i.e., a sequence of $(\hat{R}_t, s_t, a_t)$, which is then used for model training after incorporating position encoding information. The training process resembles a text generation task, where the DT predicts the output reward sum $\hat{R}$, state s, and action a simultaneously for the next position based on the preceding sequence. However, considering the spontaneous changes in state transfer and reward feedback information during actions in reinforcement learning environments, treating them as predictive outputs seems unreasonable and redundant. We assess the impact of the model's output on state s and reward sum $\hat{R}$ in optimizing the model by modifying the loss function in the DT optimization process. The two loss functions employed are:

$$L_1 = \frac{1}{n}\sum_{i=0}^{n}(a_i^{'} - a_i)^2 \tag{8}$$

$$L_2 = \frac{1}{n}\sum_{i=0}^{n}[(a_i^{'} - a_i)^2 + (s_i^{'} - s_i)^2 + (\hat{R}_i^{'} - \hat{R}_i)^2] \tag{9}$$

where $a_i^{'}$, $s_i^{'}$ and $\hat{R}_i^{'}$ are our predictions.

We analyze results within the original DT experimental environment, and detailed information about the setup will be provided in Section 4.1. We optimize the DT model using two different loss functions (L_1 and L_2), denoted as DT (L_1) and DT (L_2), respectively. We compare the two DT models using offline datasets of varying qualities (introduced in Section 4.2), calculating the total reward values obtainable from the predicted trajectories of the models after 100k training steps. The results are shown in Table 1.

Table 1. Performance results of DT with different types of loss functions.

Environment	Dataset	DT (L_1)	DT (L_2)
Walker	Medium	74.0	73.1
	Medium-Expert	108.1	108.3
	Medium-Replay	66.6	67.2
Hopper	Medium	67.6	67.0
	Medium-Expert	107.6	108.0
	Medium-Replay	82.7	78.4

Observing the experimental results, we can conclude that incorporating the predicted output of the reward sum $\hat{R}$ and environmental state information s into the loss function does not lead to an improvement in the model's predictions. Despite adapting the GPT structure used by the DT algorithm to the required input data format for training, it still outputs sequence data in the same format as the input, predicting the generation of additional redundant information $(\hat{R}, s)$. For the reinforcement learning task, our primary goal is to obtain the optimal action maximizing the reward for the intelligent agent. In this paper, we will focus on experiments related to this specific problem.

3.2. Model Formulation

3.2.1. Basic Introduction of the Model

We have devised a novel reinforcement learning sequence decision model called the Action-Translator Transformer (ATT) for the task of generating reinforcement learning sequence trajectories. This model leverages existing trajectory data to learn the relationship between the target reward and the trajectory. It learns sequences that lead to the target reward, allowing the generation of corresponding trajectories based on a user-defined target reward value. This approach aims to achieve the maximum target reward in the current environment, completing the reinforcement learning task. Similar to DT, ATT directly models trajectory data and avoids the training instability associated with bootstrapping

and iteration, eliminating issues related to MDP-based reinforcement learning errors (refer to Appendix A).

To ensure that our model focuses solely on predicting actions without providing additional environment and reward information, we draw inspiration from the field of NLP, specifically the text translation problem. We conceptualize the training process as a translation task, transforming state and reward information into the corresponding actions. As illustrated in Figure 4, we define a target reward value (the maximum reward value of the environment) based on environment-specific information or prior knowledge. This target reward value, along with the current environment state information, is inputted into our ATT model, which then predicts the action to be performed. After interacting with the environment to update the state and receive instant rewards, we update the historical trajectory information, the current target reward value, and the current state information. This process is repeated, predicting the next action until reaching a termination state or satisfying the target reward. This outlines the fundamental process of using ATT to predict optimal actions and obtain optimal trajectories.

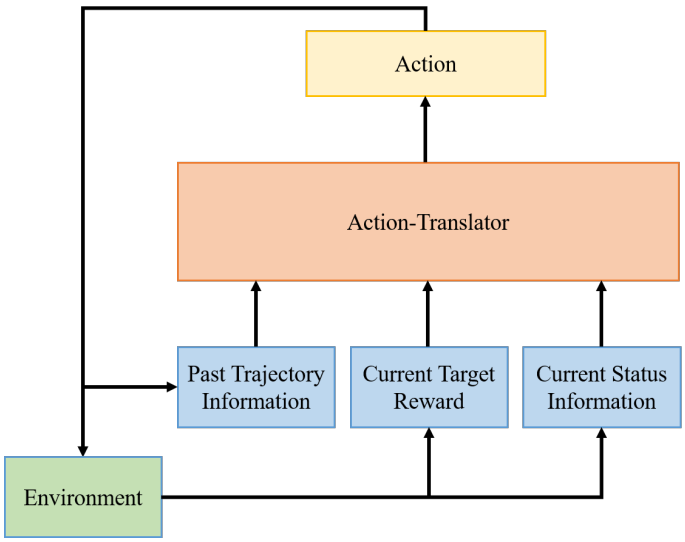

Figure 4. Schematic of the overall framework of the ATT model, abstracting the predicted actions into text translation task solving. The details of the Action-Translator are shown in Figure 5.

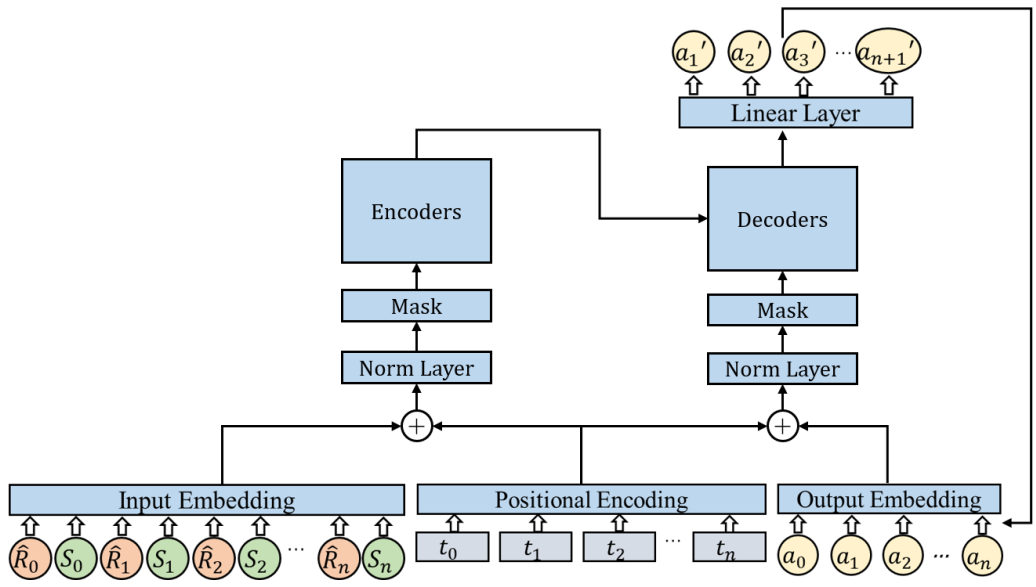

Figure 5. Action-Translator Transformer overall model architecture.

3.2.2. Overall Structure of the Model

To ensure that the model's output consists solely of optimal actions while using the environment and target reward as objective information for training, we leverage text translation task processing. We opt for the Transformer structure, encompassing both the encoder and decoder sides, as illustrated in Figure 5. To establish a connection between the reward information and context, we employ the reward treatment from DT, recalculating immediate rewards in offline data as the total reward obtained from the current moment to the end of the trajectory. Considering our dual-end structure, the intelligent agent observes the target reward value and current environment state information, serving as the input for the encoder side (i.e., as global information for the intelligent agent). The decoder side, responsible for action prediction output, undergoes different input forms during the pre-training and inference phases. The specific processes for training and inference will be detailed below. Moreover, as our model's overall structure is based on the Transformer, a substantial dataset is required to meet the model's training needs, making it primarily suitable for offline reinforcement learning tasks.

Our model follows the Sequence-to-Sequence style, comprising an encoder side and a decoder side. The encoder side consists of six encoders stacked together, receiving the encoded and masked global environment information $(s, \hat{R})$ and inputting the result into our decoder side. The decoder side also comprises six stacked decoders, primarily conducting the action prediction learning process. It accepts the encoded and masked action (a) sequence information and predicts the next action based on the observable action and global environment information input from the encoder side, completing the decision-making process for reinforcement learning. This framework represents the overall structure of the ATT model, adopting the Transformer model architecture format. It is divided into an encoder side and a decoder side, each with its corresponding encoding structure. Data are input into the encoder and decoder sides from the initial encoding module, with the final action prediction output through the linear layer in the decoder side.

3.2.3. Model Details

Target Reward: As we employ offline reinforcement learning data, traditional reinforcement learning sample data are stored in the format of (s_t, a_t, r_t). However, the Transformer model is designed to train sequence data containing contextual information. Therefore, it is necessary to establish a connection between the data and, simultaneously, segment the data into appropriate lengths for the Transformer network to facilitate learning. The representation of offline reinforcement learning trajectory data are shown in the equation below:

$$\tau = (\ldots, s_{t-1}, a_{t-1}, r_{t-1}, s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, \ldots) \tag{10}$$

where s represents the state information at a certain moment, a represents the action information at a certain moment, and r represents the instant reward information at a certain moment. Such trajectory data can only represent the relevant information at each moment, and it cannot be used as the training data for the Transformer model to learn contextual information. Therefore, we need to process the data. We retain the original forms of s and a in the trajectory data, and process r. We represent r in a new form, denoted as $\hat{R}$, calculated as follows:

$$\hat{R}_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_{end} \tag{11}$$

The information represented by $\hat{R}_t$ is the total reward value available from the current moment to the end of the trajectory. This transformation establishes a link between the data at different moments, and the trajectory history data are processed into the following form:

$$\hat{\tau} = (\hat{R}_0, s_0, a_0, \hat{R}_1, s_1, a_1, \ldots, \hat{R}_t, s_t, a_t) \tag{12}$$

Although the Transformer can handle long sequences of text, it is also sensitive to sequence length limitations. Therefore, we need to preprocess the trajectory data into segments of length K that the model can accept. In cases where the original trajectory is shorter than K, we fill the remaining positions with zeros.

Embedding: After processing, the data can meet the training requirements of the Transformer. Following the flow of a text translation task, we need to encode the processed data. We use a fully connected layer to encode the state s, action a, and target reward $\hat{R}$. Utilizing the dual-end structure of the Transformer, we concatenate $\hat{R}$ and s to form a sequence on the encoder side, encoding it as observable environmental information during action translation. On the decoder side, the sequence composed of a is encoded as the predicted output control of our model. We input the real action information as the target prediction into the decoder side, because we employ the Teacher-Forcing training method, a parallel training method of Transformer, which will be explained in detail later. Here, we only introduce the input form. Both the input-side coding layer and the output-side coding layer use a fully connected layer structure, as shown in Figure 6.

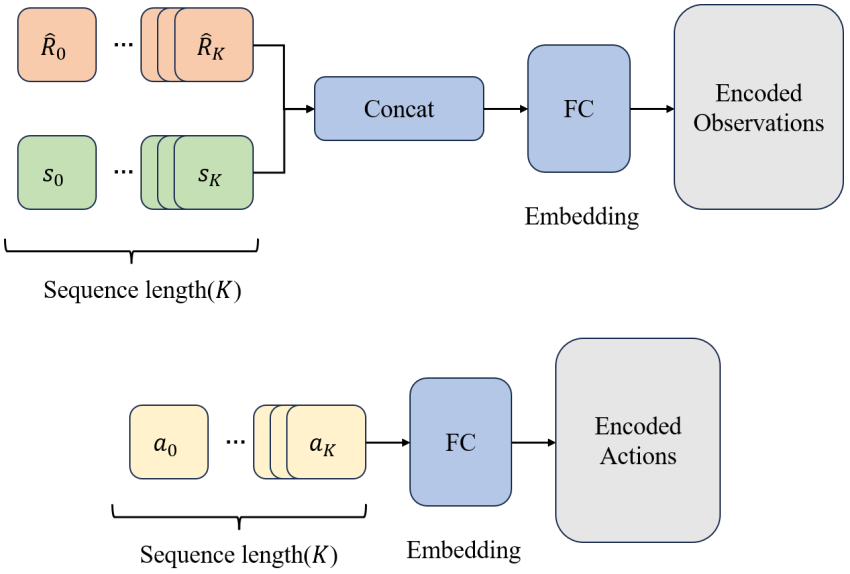

Figure 6. Encoding process of the original input in Encoders and Decoders.

Positional Encoding: During the training process of the Transformer network model, it is essential to introduce positional information to enable the model to perceive the relative positions of elements in the sequence. As our reinforcement learning trajectory data already includes temporal information, i.e., it corresponds to a specific time step, we can directly use the time step as our positional information. Considering a complete trajectory as a timeline, sequentially numbered starting from zero, we input the position encoding function to generate the positional encoding information. Our position encoding method involves creating a coding matrix that maps a certain range of numbers to unique vectors, which serve as the encoded positional information. The network structure details of the encoding part are illustrated in Figure 7. Following the encoding of positional information, we directly sum the position information encoded at each time step with the Encoder Observations and Encoder Actions obtained in the previous step. The objective of this step is to incorporate temporal information into our training data, facilitating the model in learning the causal sequence information of the entire trajectory. This encompasses understanding the distinct state transition relationships between time nodes and changes in reward information.

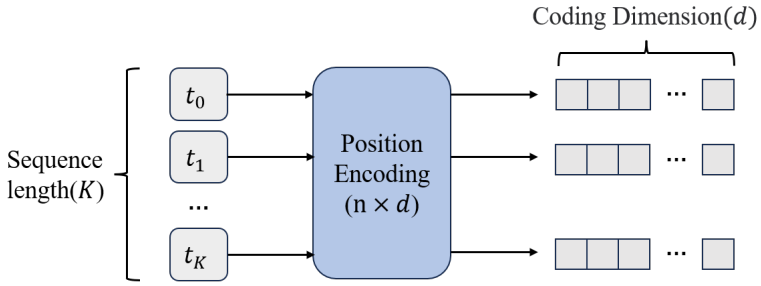

Figure 7. Flowchart of location information encoding.

Mask: After data preprocessing and the introduction of positional information, we obtain the data required to meet the training needs of the Transformer. At this stage, it is essential to consider the distinction between the model during the Training phase and the Inference phase—ensuring consistent contextual data forms. During the Training phase, using offline trajectory data, a complete trajectory is known. Therefore, the inputs on both the encoder and decoder sides are segments extracted from the entire trajectory. However, in the Inference phase of Reinforcement Learning, actions are initiated from a random state, extrapolating trajectory generation, and this process cannot observe global trajectory information. To align the tasks in the Train and Inference phases, and to cater to the lack of global trajectory information in the Inference phase, we need to introduce the Mask mechanism. Masks are applied to both the encoder and decoder sides, taking the form of upper triangular matrices. This ensures that, at a given time point, the intelligent agent can only use information from that specific moment and preceding moments to predict actions.

The input data from both Encoders and Decoders undergo Mask processing, as illustrated in Figure 8. Each sample input transforms into a square matrix, and each row of the matrix has certain information masked out. This masking is crucial for achieving parallel training using the Transformer. In the matrix, each row corresponds to a predicted output, and the predicted value in each row is the first value that has been masked out in the current row. When predicting an action at a specific time, only the real information before that point in time is used. This training approach is known as Teacher-forcing [20], a method effective in avoiding unstable network training and accelerating convergence. Employing this method allows us to simultaneously train predictions for each position in a sample, thus expediting the model's convergence.

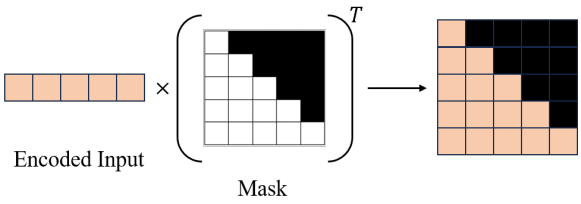

Figure 8. Mask mechanism for vectors.

The incorporation of the Mask mechanism enables our model to undergo parallel training across multi-dimensional matrices for multiple time points, expediting the training process. The training method under the Mask mechanism is shown in Figure 9. Moreover, it facilitates the alignment of task types between the training and testing phases. In the Inference phase, where the complete sequence of future actions is unknown, aligning the tasks ensures that our model performs optimally in prediction scenarios.

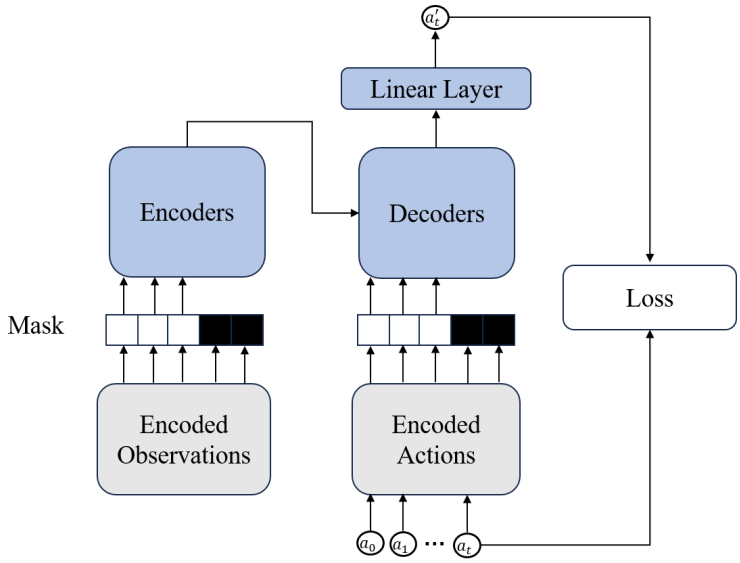

Figure 9. In the Teacher-forcing-based training process, the encoded data undergo Mask processing, marking the data after a specific point in time as invisible. The data input into the subsequent steps contains only the information before that point in time. After the encoding and decoding of the known information, and finally through the linear layer of decoding, we obtain the corresponding time point of the action a_t and the predicted value a'_t. Subsequently, we can optimize the model based on the loss of the action. The details of the loss model are shown in Equation 8.

Encoders–Decoders: The data subjected to Mask processing is subsequently input into the Encoders and Decoders, constituting the standard Transformer structure. Both Encoders and Decoders consist of several small structures stacked together, with the architectural intricacies depicted in Figure 10, following the design outlined by Vaswani et al. [6].

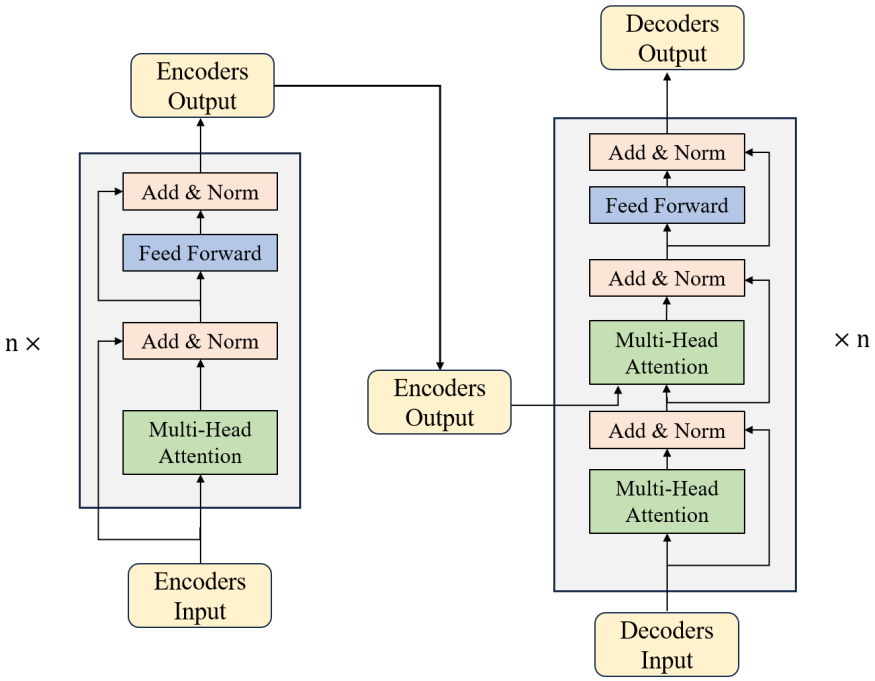

Figure 10. The internal structure of Encoders and Decoders comprises numerous smaller encoder structures. These smaller encoders encompass attention calculations and various network layer structures. The output of the last small encoder is then transmitted to each small decoder to engage in attention calculations within the encoder. Ultimately, the final output is derived from the last small decoder.

Each small Encoder incorporates a Multi-Head Attention calculation, employed to compute the similarity between the current state of the agent's environment and the state in the preceding sequence. Additionally, a Feed Forward component is included, facilitating a linear transformation that maps the data to a high-dimensional space and subsequently to a low-dimensional space, enabling the extraction of more profound features.

In Decoders, unlike in Encoders, two Multi-Head Attention computations take place. In the second computation, the environment feature information from the agent, outputted by the Encoders, serves as both Q and K in the Attention computation. This allows our model to learn the action output based on the environment information.

Action Decoding: After obtaining the data output from the Decoders, we have completed the process of action prediction. The actions at this stage are high-dimensional encoded actions that need to be decoded. In this experiment, we directly use a linear layer to transform the corresponding high-dimensional actions into the dimensions of the intelligent body's actions in the current environment, representing the final output of the predicted actions.

3.2.4. Training and Inference

The whole training process of ATT we can summarize as Algorithm 1.

Algorithm 1 ATT training, optimize network parameters based on offline trajectories

Input: Data set $D = \{(s_t, a_t, r_t)\}$
$\hat{R}_t \leftarrow r_t + r_{t+1} + r_{t+2} + \cdots + r_{end}$
Initialize weight θ, training iterations I, step size η, batch size B, the state embedding matrix $W_s \in \mathbb{R}^{(d_{\hat{R}}+d_s)\times B}$, the action embedding matrix $W_a \in \mathbb{R}^{d_a \times B}$
for $i = 1$ **to** I **do**
  Stack $(\hat{R}, s)$
  Embedding: $(\hat{R}, s)_{emb} \leftarrow W_s \times (\hat{R}, s)$
  Embedding: $a_{emb} \leftarrow W_a \times a$
  Positional Encoding
  $Encoders \leftarrow (\hat{R}, s)_{emb} + P(t)$
  $Decoders \leftarrow a_{emb} + P(t)$
  Masked
  Train
  Output a'
  $loss \leftarrow L(a', a)$
  $\theta \leftarrow \theta + \eta \nabla L(\theta)$
end for
return θ

At the beginning of training, for the offline dataset samples, firstly, r in the sample data are calculated as $\hat{R}$, and the samples are processed to a specific length. After encoding and introducing the position information, they are processed by Mask and then entered into the Transformer for training using the Teacher-forcing method. The prediction of the action a' is then compared with the original input action sequence a to calculate the loss for optimizing the model. The complete pre-training process of ATT is outlined above. The loss function used here is given by Equation (8).

The Inference process of ATT, we can summarize as Algorithm 2.

The Inference of the ATT model is mainly derived using self-regression, where for each prediction, the model's output is used as input for the next prediction. By initializing the model with the necessary information, we continuously utilize the model to predict actions and generate trajectories. Eventually, we obtain the target trajectory based on our initial settings. During this process, the inputs to the model include the cumulative target rewards, where we set the maximum possible reward value $\hat{R}$ (as the ultimate goal of reinforcement learning is to achieve the maximum reward in the current environment) based on prior knowledge or environmental information. The initial state of the environment s is obtained

directly by initializing the environment. We initialize the action a on the decoder side and input all this information into the encoder and decoder sides according to the input mode during the training process after padding operation. Specifically, $\hat{R}$ and s are input into the encoder side as global observable information and then passed to the decoder side after going through the Mask and Encoders.

Algorithm 2 ATT inference, generate full trajectories based on target rewards

```
Initializeenv()
Initialize R̂, s, a, step = 0, length, G = 0
Padding
while s not equal to end or step<length do
    Positional Encoding
    Encoders ← (R̂, s) + P(t)
    Decoders ← a + P(t)
    Masked
    Output a_next
    r_next, s_next ← env(a_next)
    R̂ ← R̂ − r_next
    G ← G + r_next
    step ← step + 1
end while
return G
```

The next action is predicted at the decoder side based on the input historical action information a and the global observable information from the encoder side. After the model outputs the next action a_{next}, we execute a_{next} in the current environment. At this point, our environment changes to s_{next} due to the agent's interaction behavior, and we obtain the instant reward r_{next}. Based on this instant reward, we update the expected reward sum $\hat{R}_t$ and then add the newly obtained a_{next}, $\hat{R}_{\text{next}}$, s_{next} to the existing sequence as the observable data for predicting the next action. This process continues, and eventually, based on our artificially set expected reward sum $\hat{R}$, we obtain a complete trajectory. Through the accumulation of instant rewards r obtained during the process, we can calculate the total reward sum of the trajectory, G, and evaluate the strengths and weaknesses of our model.

4. Experimental Validation

Because both our ATT and DT utilize sequence modeling to address offline reinforcement learning tasks, we will compare them with DT and some classical offline reinforcement learning algorithms in a specific experimental setting to analyze the strengths and weaknesses of our models.

4.1. Experimental Environment

The experiment utilizes the Mujoco [21] simulation environment within the Gym environment. This environment incorporates a comprehensive robot motion engine, providing detailed motion information for each component of the intelligent agent. It encompasses the state information of the agent, along with reward-related settings, making it suitable for training in reinforcement learning tasks. The environment offers a variety of task scenarios, and the experiment specifically involves conducting assessments in the Hopper, Walker2D, and HalfCheetah environments. The experimental scenario is illustrated in Figure 11.

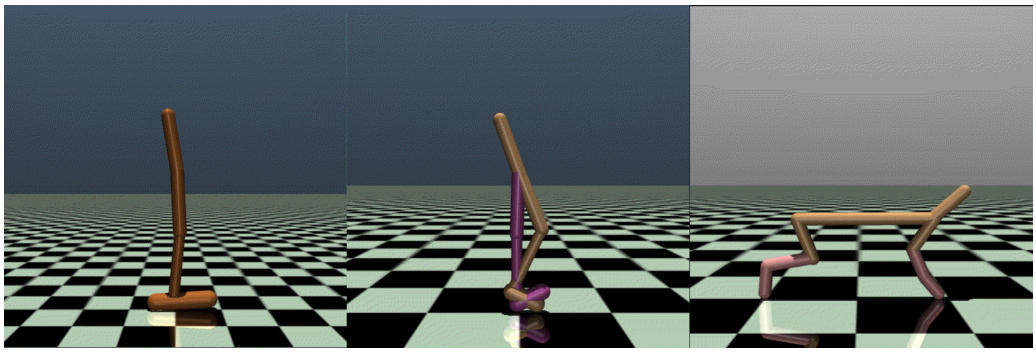

Figure 11. The Hopper, Walker, and HalfCheetah environments represent versatile experimental scenarios within a general-purpose physics engine. This engine is specifically designed to support various domains, including robotics, biomechanics, graphics and animation, machine learning, and other fields requiring precise simulation of articulated structures interacting with their environment.

Hopper: This environment features a single-legged robot with four main body parts: torso, thighs, legs, and feet. The robot's motion involves the control of three joints to execute a jumping maneuver. The state observation encompasses angular velocity, joint angles, and coordinates. The action space is defined by the control values for the three joints, and rewards are provided based on the robot maintaining an upright posture and successfully propelling forward.

Walker: In this environment, the robotic body features two complete legs and is composed of four main components: torso, thighs, legs, and feet. Unlike Hopper, the intelligent body in this scenario has two legs and two feet. The state observation space is similar to that of Hopper, while the action space expands in spatial dimension due to the increased number of joints in the intelligent body. Reward feedback is obtained when the intelligent agent maintains an upright position and successfully propels forward.

HalfCheetah: This environment involves a two-legged robot that exhibits animal-like movements and endeavors to progress within the plane defined by its legs. The state observation space encompasses the angles, angular velocities, and coordinates of the joints. Similarly, the action space includes control information for the joints. Reward feedback is provided when the intelligent agent maintains an appropriate upright posture and moves in the correct direction.

In the second segment of the experiment, we utilized the Maze2D game environment, focusing on a navigation task. The objective of the test is to discover the shortest path to the target point by integrating sub-optimal trajectories with the Offline RL algorithm. This environment comprises three game scenarios: Umaze, Medium, and Large, each featuring a maze structure illustrated in Figure 12. The primary evaluation of the model's performance revolves around its capacity to concatenate sub-optimal trajectories to efficiently find the shortest path to the target point.

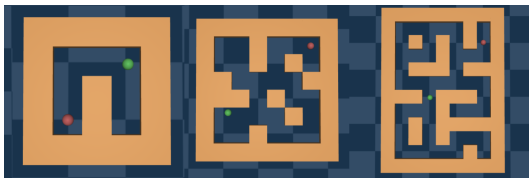

Figure 12. Umaze, Medium, and Large three maze environments, Maze2D is a navigation task. The test goal is to find the shortest path to the target point by combining the sub-optimal trajectory with the offline RL algorithm. In the illustration, the green dots represent the current position of the agent, while the red dots indicate the target points that the agent needs to reach.

4.2. Dataset

The dataset utilized in this experiment is sourced from the Python data package D4RL [22]. It encompasses the complete action trajectory of an agent in the Mujoco

environment under Gym, incorporating the state information s of the agent at a specific moment in time, the action information a taken by the agent, and the instantaneous rewards r acquired by the agent. The dataset is structured as (s_t, a_t, r_t). Within this package, various trajectory data types are available for the same game environment, originating from diverse data collection methods. For this experiment, the focus is primarily on three data types:

- Medium dataset: A strategy network is trained online using Soft Actor-Critic (SAC), and the training is prematurely terminated to collect 1M samples from this partially trained strategy.
- Medium-replay dataset: Comprising all samples recorded in the replay buffer during training until the policy achieves a "medium" performance level.
- Medium-expert dataset: This dataset introduces a "medium-expert" dataset by blending an equal amount of expert demonstration and sub-optimal data. The latter is generated either by partially trained strategies or by expanding a uniformly randomized strategy.

These three data types offer comprehensive coverage of the domains in which our model is tested, allowing for a thorough evaluation of the model's strengths and weaknesses.

Additionally, for the Maze2D environment, we trained the model using the tracks for the three scenarios stored in the D4RL dataset. The performance is then compared with the CQL and DT algorithms.

4.3. Experimental Results

We compare ATT with DT and several other key offline reinforcement learning algorithms, including CQL [23], BEAR [24], BRAC [25], AWR [26], and BC [27]. Our ATT model focuses on transformational tasks, DT is centered around generative tasks, and CQL is recognized as a top-performing MDP-based RL algorithm. The experiments are conducted in three Mujoco scenarios. For ATT, we configure both the stacked Encoder and Decoder structures with six layers, set the learning rate to 1×10^{-3}, and establish the length of receivable sequences (K) at the encoder and decoder sides to be 20. The sum of target rewards ($\hat{R}$) is determined based on historical trajectory data, with Hopper set to 3600, HalfCheetah to 12,000, and Walker to 5000. After training for $100k$ steps under identical conditions, we compare the effectiveness of various algorithms. To facilitate comparison, we normalize the results using the training results of the SAC algorithm as the expert score and those of a random strategy as the random score. The normalization formula used is as follows:

$$normalized = 100\% \times \frac{score - random}{expert - random} \tag{13}$$

The results of the experiment are shown in Table 2. The results of the DT are referenced from [11], while the results of CQL, BEAR, BRAC-v, AWR, and BC are referenced from [22].

Table 2. Comparison of ATT with other offline reinforcement learning algorithms in Mujoco.

Dataset	Environment	ATT (Ours)	DT	CQL	BEAR	BRAC-v	AWR	BC
Medium-expert	Hopper	109.8	107.6	**111.0**	96.3	0.8	27.1	76.9
	Walker	**110.2**	108.1	98.7	40.1	81.6	53.8	36.6
	HalfCheetah	**88.9**	86.8	62.4	53.4	41.9	52.7	59.5
Medium	Hopper	**68.4**	67.6	58.0	52.1	31.1	35.9	63.9
	Walker	**82.0**	74.0	79.2	59.1	81.1	17.4	77.4
	HalfCheetah	40.3	42.6	44.4	41.7	**46.3**	37.4	43.1
Medium-replay	Hopper	79.3	**82.7**	48.6	33.7	0.6	28.4	27.6
	Walker	**68.7**	66.6	26.7	19.2	0.9	15.5	36.9
	HalfCheetah	36.3	36.6	46.2	38.6	**47.7**	40.3	4.3

The bold numbers in each row represent the best performance in the corresponding environment for that row.

We compare ATT, DT, and CQL in three scenarios of Maze2D to investigate the model's ability to splicing sub-optimal trajectories. The results are shown in Table 3.

Table 3. Comparison of ATT with other offline reinforcement learning algorithms in Maze2D.

Environment	ATT	DT	CQL
Maze2D-umaze	42.2	31.0	**94.7**
Maze2D-medium	13.7	8.2	**41.8**
Maze2D-large	10.2	2.3	**49.6**

The bold numbers in each row represent the best performance in the corresponding environment for that row.

4.4. Discussion

Based on the experimental results, we conclude that our ATT model exhibits comparable performance to DT, outperforming DT in most scenarios and settings. Moreover, ATT surpasses many traditional offline reinforcement learning algorithms, demonstrating an ability to learn trajectories with superior reward payoff values. While CQL, an offline reinforcement learning algorithm based on MDP, maintains a certain advantage, it struggles with low data quality, unlike ATT and DT, which leverage sequence modeling to enhance sample utilization in offline datasets. This allows them to learn effectively, even from suboptimal offline data. These findings highlight the efficacy of using trajectory data for decision-making in offline reinforcement learning. Our transformational modeling approach in ATT yields better results compared to the generative model DT. Notably, ATT places a greater emphasis on action output, separating the sequence of environmental information from the sequence of actions. As a result, our model achieves superior performance in the realization of suboptimal trajectory splicing compared to DT.

5. Conclusions

In our work, we explore a novel application of reinforcement learning in the domain of Natural Language Processing (NLP), specifically focusing on the Text Translation Task. Drawing inspiration from the original work on the Dual-Transformer (DT), we devise a unique approach to data segmentation and encoding, tailoring it to meet the requirements of bipartite training for the Transformer model in the context of offline reinforcement learning. The designed ATT model structure incorporates a comprehensive set of Train and Inference processes, ensuring consistent data distribution across different stages of the task for effective training. By utilizing an input method that considers both the environment and target rewards and predicts action values, our model aligns with the reinforcement learning objective of identifying optimal actions and generally outperforms DT in various tasks.

Given that the combination of language modeling and reinforcement learning is still in its early stages, and NLP is rapidly evolving with the emergence of more efficient models, the potential for RL-NLP integration remains vast. In light of our model's future research direction, we identify key aspects for consideration:

- **Target Reward Setting:** In both ATT and DT, we adopt the supervised learning task training approach, requiring the calculation of a target reward value as the trajectory label for offline dataset training. Fluctuations in experimental results may be attributed to the accurate setting of reward values. Future research should explore methods to precisely set reward values or update them dynamically during training.
- **Efficient Language Model Training:** The language model, serving as the foundational training architecture, demands a substantial amount of datasets for effective training. While reinforcement learning can generate trajectory data through interaction, this method is not applicable to online reinforcement learning due to efficiency concerns. Investigating how to train the language model in an online interactive manner becomes a focal point for future work, expanding the model's usability.
- **Exploration of Different NLP Models:** The field of NLP boasts numerous powerful models, each with distinct characteristics and adaptive capabilities. Future endeavors

can involve experimenting with other high-performing NLP models, aligning them with various reinforcement learning tasks, and further exploring the potential of combining these technologies.

Author Contributions: Methodology, software, validation and writing—original draft preparation, J.L.; writing—review and editing, N.X. and T.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Key R&D Program of China (Grant No. 2022YFB3104600), and by Chengdu Science and Technology Project (2019-YF08-00285-GX), and by the National Natural Science Foundation of China under Grant NO. 61976156, and by Intelligent Terminal Key Laboratory of SiChuan Province under Grant SCITLAB-30005.

Data Availability Statement: The dataset used in this study is the publicly available dataset D4RL, which can be found at the following website: https://github.com/Farama-Foundation/D4RL, accessed on 7 January 2024.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Appendix A. Error Analysis of Reinforcement Learning Based on MDP

Take DQN, a classical deep reinforcement learning algorithm, as an example. In order to adapt to the training scenario of higher dimensions of reinforcement learning, DQN is an algorithm that introduces neural networks into reinforcement learning. It uses neural networks to approximate fit the value function in reinforcement learning. TD algorithm update goal is:

$$y_t = r_t + \max_a Q(s_{t+1}, a; W) \tag{A1}$$

where r_t is the immediate reward at time t, Q is the action value function, s_{t+1} is the state at the next time, and W is the neural network parameter. The Gradient Descent method is used to update the weight of the neural network:

$$W \leftarrow W - \alpha \cdot (Q(s_t, a_t; W) - y_t) \cdot \frac{\partial Q(s_t, a_t; W)}{\partial W} \tag{A2}$$

where α is the update step. The error in the updating process mainly comes from two aspects: the use of maximization operators and the problems caused by bootstrap.

Appendix A.1. Maximization

For n real numbers: $x_1, x_2, \ldots, x_n$, we add noise with a mean of 0 to these n real numbers to obtain: $Q_1, Q_2, \ldots, Q_n$. Because the mean value of noise is 0, it does not change the mean value:

$$E[mean_i(Q_i)] = E[mean_i(x_i)] \tag{A3}$$

In the above formula, *mean* represents the mean function, and $mean_i(x_i)$ signifies taking the mean of the entire set of x_i. Adding noise with a mean of 0 affects the maximum value of the new real sequence:

$$E[max_i(Q_i)] \geq E[max_i(x_i)] \tag{A4}$$

In the above formula, *max* represents the maximum function, and $max_i(x_i)$ signifies taking the maximum value over the entire set of x_i. For the observed action value: $x(a_1), x(a_2), \ldots, x(a_n)$, the motion value estimate with noise is obtained by DQN: $Q(s_1, a_1; W)$, $Q(s_2, a_2; W), \ldots, Q(s_n, a_n; W)$. We assume that it is an unbiased estimate:

$$mean_a x(a) = mean_a Q(s, a; W) \tag{A5}$$

the following overestimation problem will arise:

$$q = \max_a Q(s, a; W) \geq \max_a x(a) \tag{A6}$$

Therefore, the TD update method will cause overestimation of the target value and introduce errors.

Appendix A.2. Bootstrapping

Suppose that the DQN algorithm has overestimated the action value, then the action value of $t + 1$ moment is overestimated as:

$$Q(s_{t+1}, a; W) \tag{A7}$$

then the optimal action value at time $t + 1$ is further overestimated:

$$q_{t+1} = \max_a Q(s_{t+1}, a; W) \tag{A8}$$

When the optimal action value at time $t + 1$ is used to update the Q network, the overestimation of DQN is further aggravated.

References

1. Lu, Y.; Li, W. Techniques and Paradigms in Modern Game AI Systems. *Algorithms* **2022**, *15*, 282. [CrossRef]
2. Prudencio, R.F.; Maximo, M.R.O.A.; Colombini, E.L. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *IEEE Trans. Neural Netw. Learn. Syst.* **2023**. [CrossRef] [PubMed]
3. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Minneapolis, MN, USA, 2–7 June 2019; pp. 4171–4186.
4. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. In Proceedings of the Advances in Neural Information Processing Systems 33, Online, 6–12 December 2020; pp. 1877–1901.
5. OpenAI. GPT-4 Technical Report. *arXiv* **2023**, arXiv:2303.08774.
6. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In Proceedings of the Advances in Neural Information Processing Systems 30, Los Angeles, CA, USA, 4–9 December 2017.
7. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
8. Dey, R.; Salem, F.M. Gate-variants of gated recurrent unit (GRU) neural networks. In Proceedings of the 2017 IEEE 60th International Midwest Symposium on Circuits And Systems (MWSCAS), Boston, MA, USA, 6–9 August 2017; pp. 1597–1600.
9. Parisotto, E.; Song, F.; Rae, J.; Pascanu, R.; Gulcehre, C.; Jayakumar, S.; Jaderberg, M.; Kaufman, R.L.; Clark, A.; Noury, S.; et al. Stabilizing transformers for reinforcement learning. In Proceedings of the 37th International Conference on Machine Learning, Online, 13–18 July 2020; pp. 7487–7498.
10. Banino, A.; Badia, A.P.; Walker, J.; Scholtes, T.; Mitrovic, J.; Blundell, C. Coberl: Contrastive bert for reinforcement learning. In Proceedings of the Tenth International Conference on Learning Representations (ICLR), Virtual Event, 25–29 April 2022.
11. Chen, L.; Lu, K.; Rajeswaran, A.; Lee, K.; Grover, A.; Laskin, M.; Abbeel, P.; Srinivas, A.; Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. In Proceedings of the Advances in Neural Information Processing Systems 34, Online, 6–14 December 2021; pp. 15084–15097.
12. Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; Kuksa, P. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.* **2011**, *12*, 2493–2537.
13. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
14. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. In Proceedings of the Advances in Neural Information Processing Systems 26, Lake Tahoe, NV, USA, 5–10 December 2013.
15. Sutton, R.S.; McAllester, D.; Singh, S.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In Proceedings of the Advances in Neural Information Processing Systems 12, Denver, CO, USA, 29 November–4 December 1999.
16. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 20–22 June 2016; pp. 1928–1937.
17. Mishra, N.; Rohaninejad, M.; Chen, X.; Abbeel, P. A simple neural attentive meta-learner. In Proceedings of the 6th International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, 30 April–3 May 2018.

18. Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J.; Le, Q.V.; Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. In Proceedings of the 57th Conference of the Association for Computational Linguistics (ACL), Florence, Italy, 28 July–2 August 2019; pp. 2978–2988.
19. Kumar, S.; Parker, J.; Naderian, P. Adaptive transformers in RL. *arXiv* **2020**, arXiv:2004.03761.
20. Goodman, S.; Ding, N.; Soricut, R. TeaForN: Teacher-forcing with n-grams. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, Online, 16–20 November 2020; pp. 8704–8717.
21. Todorov, E.; Erez, T.; Tassa, Y. Mujoco: A physics engine for model-based control. In Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura-Algarve, Portugal, 7–12 October 2012; pp. 5026–5033.
22. Fu, J.; Kumar, A.; Nachum, O.; Tucker, G.; Levine, S. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv* **2020**, arXiv:2004.07219.
23. Kumar, A.; Zhou, A.; Tucker, G.; Levine, S. Conservative q-learning for offline reinforcement learning. In Proceedings of the Advances in Neural Information Processing Systems 33, Online, 6–12 December 2020; pp. 1179–1191.
24. Kumar, A.; Fu, J.; Tucker, G.; Levine, S. Stabilizing off-policy q-learning via bootstrapping error reduction. In Proceedings of the Advances in Neural Information Processing Systems 32, Vancouver, BC, Canada, 8–14 December 2019; pp. 11761–11771.
25. Wu, Y.; Tucker, G.; Nachum, O. Behavior regularized offline reinforcement learning. In Proceedings of the Asian Conference on Machine Learning (ACML 2021), Virtual Event, 17–19 November 2021; pp. 204–219.
26. Peng, X.; Kumar, A.; Zhang, G.; Levine, S. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv* **2019**, arXiv:1910.00177.
27. Torabi, F.; Warnell, G.; Stone, P. Behavioral cloning from observation. In Proceedings of the 27th International Joint Conference on Artificial IntelligenceJuly 2018, Stockholm, Sweden, 13–19 July 2018; pp. 4950–4957.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.