Transformer for Trajectory Optimization: RL as A Sequence Modelling Problem

Alfonsus Rodriques Rendy
School of Data Science
The Chinese University of Hong Kong, Shenzhen
121040014@link.cuhk.edu.cn

Abstract

The transformer architecture has demonstrated remarkable capabilities across various tasks, owing to its ability to handle long sequences effectively. Inspired by its success, reinforcement learning (RL) has increasingly been formulated as a sequence modeling problem, where the unique features of transformers make them a natural fit. The sequence modeling approach in RL has gained popularity due to its effectiveness in addressing challenges such as credit assignment through bootstrapping and improving generalization, thereby enabling agents to adapt more effectively to new tasks. This paper provides an overview of the development of RL algorithms with trajectory optimization, discusses the strengths and weaknesses of these approaches, and outlines the current challenges they face.

1 Introduction

Reinforcement Learning (RL) has emerged as a subfield of artificial intelligence focused on building systems capable of sequential decision-making. Traditional reinforcement learning methods rely on abstractions of state-value or action-value functions to model the dynamics of an environment, using algorithms like dynamic programming to compute optimal policies. Model-free methods were later developed to address challenges in complex environments where explicitly modeling the environment is infeasible. These methods approximate value functions or policies directly without relying on an explicit model [1]. While they have been successful for simple tasks, they still face challenges in handling long-term dependencies (credit assignment) [2] and generalization [3].

Transformers, particularly in the context of large language models (LLMs) in natural language processing, have demonstrated exceptional abilities to capture long-range dependencies in sequential data [4]. This success has motivated researchers to explore the application of transformers in RL, marking a paradigm shift from functional approximations (mapping states to actions) to trajectory optimization. By framing RL as a sequence modeling problem, the goal shifts to predicting the next state, action, and reward in a trajectory, conditioned not only on the current state but also on a broader context of prior trajectory data [5, 6].

This paper highlights the recent paradigm shift in reinforcement learning (RL), framing it as a sequence modeling problem using Transformers, particularly in offline RL settings. In Section 3, we begin by conceptualizing RL as sequence modeling and explain how Transformers can address this via behavioral cloning. This section also examines various setups, including pre-training objectives. Section 4 explores how applying Transformers in RL can enable the development of meta-RL agents that generalize across multiple tasks within the same domain. In Section 5, we review recent methods for training offline RL agents through learning trajectory histories, emphasizing their ability to facilitate incremental in-context learning. Finally, the paper addresses the limitations of trajectory optimization in RL and proposes directions for future research.

2 Preliminaries

Markov Decision Process We start from MDP, a mathematical model for sequential decision problem. Formally, a fully-observable MDP is denoted as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \rho_0, \mathcal{R}, \gamma)$ with $\mathcal{S}$ as the state space, $\mathcal{A}$ as the action space, $\mathcal{T}(s_{t+1}|s_t, a_t)$ as the transition probability to s_{t+1} conditioned on current state and action (s_t, a_t), $R(s_t, a_t)$ as the reward value conditioned on current state and action, $\gamma \in (0, 1]$ be the reward discount factor, and ρ_0 be the initial distribution. However, some problems are partially-observable (POMDP). In this type of problems, instead of using state $s \in \mathcal{S}$, the agent take observations $o \in \mathcal{O}$ as an input which only provide partial information about the state of the world.

Using the MDP setup, we denote experience of the agent as trajectory $\tau = (s_i, a_i, r_i)_{i=1}^{t}$ and the action selection rule as policy $\pi(a_t|s_t)$. The objective of RL algorithm is to find the optimal policy $\pi^*(a \in \mathcal{A}|s \in \mathcal{S})$ that maximizes the expected total reward achieved through actions from the policy:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi} \left[\sum_{t=1}^{T} \gamma^{t-1} r_t \right]$$

Transformer The transformer architecture proposed by Vaswani et al. consists of an encoder and a decoder component which are built on top of stacked self-attention and feedforward network [4]. The self-attention is the core component where input tokens are mapped to query, key and value vector (q, k, v) which are used to compute the attention map, the score between each pair of input vectors

$$\text{Attention(Q, K, V} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

Transformer is enhanced with positional encoding to encode the input sequence ordering, multi-head attention, and residual connections to allow more flow of information from input to output. The model is pre-trained with self-supervised manner to predict the next token sequence conditioned on previous token $p(t_i|t_{j<i})$.

3 Conditioned Behavioral Cloning

3.1 Trajectory Optimization: RL as a Sequence Model

Conditioned behavioral cloning (conditioned BC) is a method within trajectory optimization that reframes reinforcement learning (RL) as a conditional sequence modeling problem. Instead of directly learning a policy, the model learns to predict next actions (sometimes with next states and rewards) based on a sequence of past states, actions, and rewards. The model is trained on offline datasets of expert trajectories.

Pretraining Objective	**Loss Function**	**Previous Works**
Next action	$-\log P_\theta(a_t|\tau_{0:t-1}, s_t)$	
Reward-conditioned action	$-\log P_\theta(a_t|\tau_{0:t-1}, s_t, R_t)$	Chen et al.[6]
Future value and reward	$-\log P_\theta(a_t, \hat{R}_t, r_t|\tau_{0:t-1}, s_t)$	Lee et al. [7]
Goal-conditioned action	$-\log P_\theta(a_t|\tau_{0:t-1}, s_t, G_{t+i})$	
Future-conditioned action	$-\log P_\theta(a_t|\tau_{0:t-1}, s_t, z)$	Xie et al. [8]
Forward dynamics	$-\log P_\theta(s_t|\tau_{0:t-1})$	
Inverse dynamics	$-\log P_\theta(a_t|s_t, s_{t+1})$	Sun et al. [9]

Table 1: Pretraining objectives and their corresponding loss functions summarized by Liu et al. [10]

The core component of trajectory optimization involves using trajectory data as sequence modeling for decision-making. Specifically, during the training process, we are given N sequences of trajectories $\tau = \{(s_i, a_i, r_i)_{i=1}^{T}\}^N$. The trajectory optimization approach for offline reinforcement learning aims to learn a function that captures the sequential structure of the trajectory. Using transformers, this can be achieved through self-supervised learning. The formulation of the training objective can vary based on what is being predicted and what it is conditioned on, as shown in Table 1. The general algorithm for training a transformer for trajectory optimization is as follows:

Algorithm 1 Training Transformer for Trajectory Optimization, modified from [11]

```
Input: Trajectory data $\{\tau_n\}_{n=1}^N$, Initial Parameters $\theta$

procedure TRAINING($\tau$)
    for $i = 1, 2, \cdots, epochs$ do
        for $x_n \in batch(\tau)$ do
            $T \leftarrow \text{length}(x_n)$
            $P_\theta \leftarrow \text{Transformer}(x_n|\theta)$
            $\mathcal{L}(\theta) = \frac{1}{T}\sum_{t=1}^{T} \log P_\theta(x_n[t+1], t)$
            $\theta \leftarrow \theta - \alpha\nabla_\theta \mathcal{L}(\theta)$
    return $\theta$
```

Two pioneering works on this approach are decision transformer (DT) and trajectory transformer (TT). The Decision Transformer (DT) method can be considered model-free, as it predicts the next action without relying on the system's dynamics. In contrast, the Trajectory Transformer (TT) is model-based, as it first models the transition dynamics and performs planning using beam search to maximize rewards based on the learned transition model. [12]. The following subsection will discuss in detail the algorithms implemented in DT and TT and introduce the limitations of conditioned behavior cloning approach.

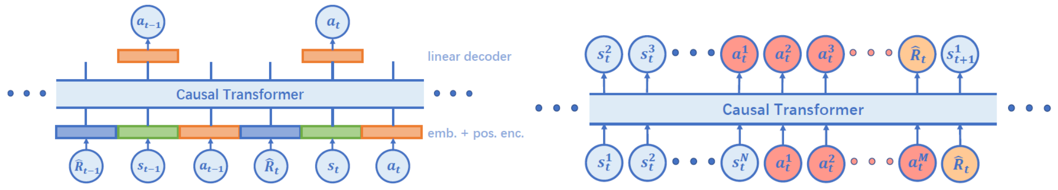

Figure 1: Comparison of architecture between DT and TT (a) DT takes states, actions, and RTGs as input and output next action (b) TT takes current dimension of state, actions, and RTGs, and output the next dimension. Illustration adapted from [12]

3.2 Decision Transformer

Algorithm 2 Decision Transformer Algorithm (Continuous Action)

```
Input: Trajectory data $\tau = \{(\bar{R}_t, s_t, a_t, t)_{t=1}^T\}^N$
procedure TRAINING($\tau$)
    Initialize parameters $\theta$
    for $i = 1, \cdots, N_{epochs}$ do
        for $(\hat{R}_t, s_t, a_t, t) \in batch(\tau)$ do
            $\hat{a}_t \leftarrow \text{DecisionTransformer}(\hat{R}_t, s_t, a_t, t, \theta)$
            $\text{loss}(\theta) = \frac{1}{T}\sum_{t=1}^{T}(\hat{a}_t - a_t)^2$
            $\theta \leftarrow \theta - \alpha\nabla_\theta \text{loss}(\theta)$
    return $\theta$
```

DT is a method that formulates RL as a reward-conditioned next action prediction. [6]. Specifically, given a trajectory

$$\tau = (\bar{R}_1, s_1, a_2, \bar{R}_2, s_2, a_2, \cdots, \hat{R}_T, s_T, a_T)$$

where $\bar{R}_t = \sum_{t'=t}^{T} r_{t'}$ be the return-to-go at timestep t, we want to train a model that predict the next action a_t at each timestep t.

During training, DT samples batches of trajectories and minimizes the cross-entropy loss for discrete actions or the mean squared error loss for continuous actions (as shown in Algorithm 5 for continuous actions). During evaluation, predictions are made online: DT selects an action, executes it in the environment, and observes the resulting state and reward.

3.3 Trajectory Transformer

Not long after DT was proposed, Janner et al. introduced a similar algorithm called the Trajectory Transformer (TT) [5]. What differentiates TT from DT is how TT models the transitions in the trajectory by discretizing each dimension independently and formulating the trajectory as

$$\tau = (s_t^1, s_t 2, \cdots, s_t^N, a_t^1, a_t^2, \cdots, a_t^M, r_t)_{t=1}^T$$

for action $\mathcal{A} \in \mathbb{R}^N$ and state $\mathcal{S} \in \mathbb{R}^M$.

TT is trained in an autoregressive manner by maximizing the log-likelihood of the next token given the previous tokens in the sequence, using the following loss function.

$$\mathcal{L}(\tau) = \sum_{t=1}^{T} \left(\sum_{i=1}^{N} \log P_\theta(s_t^i | \mathbf{s}_t^{<i}, \tau_{<t}) + \sum_{i=1}^{M} \log P_\theta(a_t^i | \mathbf{a}_t^{<i}, \mathbf{s}_t, \tau_{<t}) + \log P_\theta(r_t | \mathbf{a_t}, \mathbf{s_t}, \tau_{<t}) \right)$$

where θ is the Trajectory Transformer parameters, $\tau_{<t}$ be the trajectory up to timestep $t-1$, $s_t^{<i}$ and $a_t^{<i}$ be the state and action dimensions up to dimension $i-1$ respectively.

The Transformer model enables us to model the distribution of trajectories effectively. Planning is performed by sampling trajectories using beam search. In an imitation learning setup, the objective is to replicate expert demonstrations. To achieve this, the Trajectory Transformer (TT) utilizes its trajectory modeling capability to predict the most probable action sequence based on a history of states and actions. In offline reinforcement learning (RL), the aim is to maximize the reward using a dataset of past experiences. Consequently, TT incorporates reward information into the beam search process, prioritizing sequences with high cumulative rewards and reward-to-go estimates.

Algorithm 3 Trajectory Transformer Planning with Beam Search

```
Input: Trajectory history $\mathbf{x}$, vocabulary $\mathcal{V}$, sequence length $T$, beam width $B$
Output: Optimal Trajectory $\hat{\mathbf{y}}$
procedure BeamSearch
    Initialize $Y_0 = \{()\}$
    for $t = 1, \cdots, T$ do
        $\mathcal{C}_t \leftarrow \{\mathbf{y}_{t-1} \circ y | \mathbf{y}_{t-1} \in Y_{t-1}, y \in \mathcal{V}\}$
        $Y_t \leftarrow \arg\max_{Y \subseteq \mathcal{C}_t, |Y|=B} \log P_\theta(Y|\mathbf{x})$
    $\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in Y_T} \log P_\theta(\mathbf{y}, \mathbf{x})$
    return $\hat{\mathbf{y}}$
```

3.4 Performance Evaluation and Limitations

Behavior Cloning Performance When comparing the Decision Transformer (DT) and Trajectory Transformer (TT) for behavioral cloning, the TT demonstrates superior performance due to its explicit focus on modeling trajectories and its effective use of beam search for planning. When trained on datasets collected by expert policies, TT, using beam search, successfully replicates expert behavior [5]. This strong performance highlights TT's strength as a model-based approach for behavior cloning, surpassing the capabilities of traditional behavior cloning methods that rely on single-step action predictions.

In contrast, DT's primary focus is on conditional sequence modeling, aiming to generate actions based on desired returns. While its training process might appear similar to behavior cloning, it's not specifically designed for pure imitation. The result suggests that DT's performance in behavioral cloning is generally weaker compared to dedicated behavioral cloning methods and other offline RL algorithms. For instance, in evaluations on Atari and OpenAI Gym benchmarks, DT was outperformed by or showed comparable performance to behavior cloning in most cases [6].

Rewards Sparsity When tested in environment with sparse rewards, DT exhibits more robustness, outperforming traditional TD learning algorithms that struggle with credit assignment. Its ability to condition on returns-to-go helps guide action selection even when rewards are infrequent. However, in extremely sparse settings like the AntMaze, its performance can be limited [6].

On the other hand, sparse reward environments pose a challenge for TT. Relying solely on reward-to-go estimates for planning may lead to suboptimal results in such environments. One solution is to combine TT with learned Q-functions as a search heuristic which significantly improves performance in sparse reward tasks like AntMaze navigation [5].

Expert Trajectory Data Both the Decision Transformer (DT) and Trajectory Transformer (TT) exhibit impressive capabilities in various RL tasks, but they share a significant limitation: a reliance on expert trajectory data for effective learning. This dependence on high-quality demonstrations presents some challenges as expert data are scarce. Some expert demonstrations may also contain subpotimal trajectory. Training DT or TT on such data can lead to the model learning and replicating these flaws, resulting in less effective policies [13].

Generalization To New Task While DT and TT can generalize to new tasks within the same distribution as the training data, they often struggle to extrapolate to entirely novel scenarios or tasks with significantly different dynamics. This limitation highlights the need for mechanisms that enable the models to leverage prior knowledge and adapt to unseen situations without relying solely on expert demonstrations.

4 Generalization: Trajectory Optimization For Meta-Reinforcement Learning

Traditional RL algorithms often struggle with generalization, as they tend to overfit to the specific tasks they are trained on. This limitation also holds to DT and TT as both are fundamentally supervised learning approaches that rely heavily on the quality, quantity, and distribution of the trajectory data they are trained on. This reliance on data distribution creates challenges when DT and TT are applied to tasks or environments that deviate significantly from their training data. They might exhibit reduced performance, struggle to adapt to new dynamics, or fail to discover optimal solutions not present in the demonstrations.

Several efforts, like Multi-Game DT [7], PromptDT [14], and CMT [15], attempt to address these limitations by exploring alternative training paradigms, introducing mechanisms for few-shot adaptation, or leveraging pre-training on large, diverse datasets to enhance generalization capabilities.

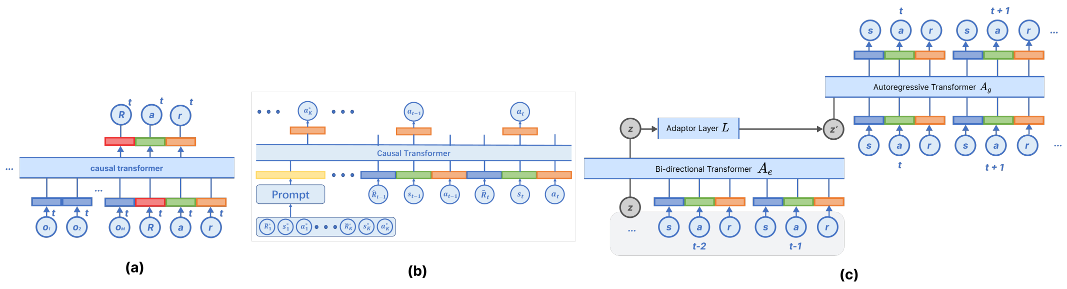

Figure 2: Trajectory Optimization for Meta-RL (a) Multi-Game DT (b) PromptDT (c) CMT. Illustration inspired by [12]

4.1 Multi-Game Decision Transformer

Multi-Game DT (MGDT) aims to train a model that predict actions that maximize the total future return on all the environments that they include in the training data [7]. They formulate the trajectory sequence as

$$\tau = (\mathbf{o}_1^t, \cdots, \mathbf{o}_M^t, \hat{R}^t, a^t, r^t)$$

for each timestep t, let M be the number of patches per observation. The ordering of $\mathbf{o}$ and $\hat{R}$ distinguishes it from traditional DT, as Multi-Game DT enables predicting the return distribution and sampling from it instead of relying on the return provided by user input.

During inference, instead of directly sampling returns from the model's predicted distribution $P_\Theta(\hat{R}t|\tau^{<t}, \mathbf{o}1 : M^t)$, MGDT modifies the sampling process to favor expert-level returns. The expert probability is defined to be proportional to the future return, with an inverse temperature κ:

$$P(\text{expert}^t|\hat{R}^t, \tau^{<t}, \mathbf{o}^t_{1:M}) := \exp(\kappa \hat{R}^t)$$

The expert probability is incorporated into the return sampling distribution:

$$\log P(\hat{R}^t|\text{expert}^t, \tau^{<t}, \mathbf{o}^t_{1:M}) = \log P_\Theta(\hat{R}^t|\tau^{<t}, \mathbf{o}^t_{1:M}) + \kappa \frac{(\hat{R}^t - \hat{R}^{low})}{(\hat{R}^{high} - \hat{R}^{low})}$$

It then sample actions based on $P_\theta(a^t|R^t)$.

4.2 PromptDT: Prompt-Based Decision Transformer

PromptDT proposes offline few-shot RL paradigm. Their motivation is to achieve generalization wiyhout finetuning or further gradient updates [14]. The core idea behind PromptDT is to utilize a "trajectory prompt" to guide the Decision Transformer (DT) model towards adapting to new, unseen tasks effectively with minimal training data.

Algorithm 4 PromptDT Training and Evaluation

```
Input: Tasks $\mathcal{T}^{train} \cup \mathcal{T}^{test}$, Demonstrations $\mathcal{P}$
procedure TRAINING($\mathcal{T}^{train}, \mathcal{P}$)
    Initialize parameters $\theta$
    for $i = 1, \cdots, N_{epochs}$ do
        for Task $\mathcal{T}_i \in \mathcal{T}^{train}$ do
            for $m = 1$ to $M$ batch-size do
                Sample trajectory $\tau_{i,m}$ of length $K$ from $\mathcal{D}_i$
                Sample prompt $\tau^*_{i,m}$
                $\tau^{input}_{i,m} = (\tau^*_{i,m}, \tau_{i,m})$
            Batch $\mathcal{B} = \{\{\tau^{input}_{i,m}\}^M_{m=1}\}^{T^{train}}_{i=1}$
            $\hat{a} = \text{DecisionTransformer}(\tau^{input}|\theta) \qquad \forall \tau^{input} \in \mathcal{B}$
            $\mathcal{L}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{\tau^{input} \in \mathcal{B}} (\hat{a} - a)^2$
            $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$
    return $\theta$

procedure EVALUATION($\mathcal{T}^{test}, \mathcal{P}, G^*$)
    for Task $\mathcal{T}_i \in \mathcal{T}^{test}$ do
        $\tau = \{\}, g = G^*$                                   ▷ $G^*$ target return
        Sample prompt $\tau^*$
        for $t \leq T$ do
            $a = \text{DecisionTransformer}((\tau^*, \tau)|\theta)[-1]$
            Step environment and observe $s^{t+1}, r$. $g \leftarrow g - r$
            Append $\tau \leftarrow \{\tau, [s, a, g]\}$
```

Specifically, they assume the existence of few-shot demonstration $\mathcal{P}_i$ for each task $\mathcal{T}_i \in \mathcal{T}^{train} \cup \mathcal{T}^{test}$. They introduce trajectory prompt, composed of short segments of expert demonstrations for the new target task which provide task-specific information:

$$\tau^*_i = (\hat{R}^*_1, s^*_1, a^*_1, \cdots, \hat{R}^*_{K*}, s^*_{K*}, a^*_{K*})$$

with K^* be the number of environment steps stored in the prompts. The trajectory prompts are sampled from the few-shot demonstration dataset $\mathcal{P}_i$ stochastically. It consisted of J trajectory segments each with length H and $K^* = JH$.

For the training and evaluation, it modifies the DT input by prepending the trajectory prompt to the recent trajectory history

$$\tau^{input} = (\tau_i^*, \tau_i)$$

with the rest of the procedure similar to that of DT.

4.3 CMT: Contextual Meta Transformer

Contextual Meta Transformer (CMT) is a novel offline RL algorithm inspired by prompt tuning in NLP. It offers a pretraining and prompt-tuning paradigm to address the challenges of generalization and multi-task learning [15].

CMT follows the same prompt-augmented trajectory generation approach as PromptDT. The key difference lies in the prompts used: while PromptDT uses prompts sampled only from demonstration data, CMT learns the prompts using a bi-directional transformer to obtain a latent representation that encodes the trajectory prompt.

CMT introduces two components: a trajectory encoder A_e, a bi-directional transformer with parameters θ, which takes a history trajectory as input and outputs the policy prompt $z_\tau = A_e(\tau|\theta)$; and an autoregressive generator A_g with parameters ϕ, which predicts the next token based on the previous history trajectory and the policy prompt, defined as $\tau_{t+1} = A_g(.|z_\tau, \tau_{<t}; \phi)$.

The training process is divided into two stages: the representation stage and the improvement stage. In the **representation stage**, CMT uses two loss terms combined as $\mathcal{L} = \mathcal{L}_1 + \gamma\mathcal{L}_2$, where γ is the contrastive loss coefficient. The first loss, $\mathcal{L}_1$, is a supervised loss aimed at reconstructing the entire trajectory:

$$\mathcal{L}_1(\tau; \phi, \theta) = \sum_{t=0}^{T-1} \big(\mathcal{D}(s_t, P(\tau_{<t}; \phi, \theta)) + \mathcal{D}(a_t, \pi(s_t, \tau_{<t}; \phi, \theta)) + \mathcal{D}(r_t, R(a_t, s_t, \tau_{<t}; \phi, \theta))\big)$$

It uses distance metrics $\mathcal{D}$, adopting MSE loss for continuous deterministic outputs and cross-entropy loss for stochastic predictions. Additionally, the loss consists of three terms, each evaluating the prediction of the policy $\pi(a|s)$, the dynamics model $P(s'|s, a)$, and the reward function $R(s, a)$.

The second loss employs self-supervised learning to constrain the distance between prompts derived from similar trajectories. This encourages new behaviors to remain close to prior ones while maximizing the reward. The loss is formulated using the InfoNCE contrastive loss:

$$\mathcal{L}_2(\tau_q, \{\tau_i\}_{i=1}^K; \theta) = -\log \frac{\exp(A_e(\tau_q; \theta) \cdot A_e(\tau_+; \theta)/\alpha)}{\sum_{i=1}^k \exp(A_e(\tau_q; \theta) \cdot A_e(\tau_i; \theta)/\alpha)} = -\log \frac{\exp(z_q \cdot z_+/\alpha)}{\sum_{i=1}^k \exp(z_q \cdot z_i/\alpha)}$$

Here, α represents the temperature coefficient, and z_q denotes the encoded prompt derived from the trajectory τ_q. Additionally, a batch of K policy prompts $z_i i = 1^K$ is encoded from a set of trajectories $\tau_i i = 1^K$ sampled from an offline dataset. This batch includes $K - 1$ negative samples z_- and one positive sample z_+. The auxiliary loss defines positive sample pairs as those derived from the same trajectory, while negative sample pairs are derived from different trajectories.

In the **improvement stage**, instead of directly modifying the pretrained model, CMT learns a prompt that effectively guides the model to generate the desired actions. This prompt acts as a "context" that steers the model's policy towards achieving higher rewards.

To facilitate prompt tuning, the pretrained model is frozen as $\bar{\theta}$ and $\bar{\phi}$. An adaptor layer L is trained to obtain the new policy prompt z' using the following loss function:

$$\mathcal{L}_3(\tau; \xi) = \sum_{t=0}^{T-1} D\left(\hat{a}_t, \pi\left(s_t, \tau_{<t}; \bar{\phi}, \bar{\theta}, \xi\right)\right) + \beta\left(z - L(z; \xi)\right)^2,$$

where the first term minimizes the divergence between the predicted actions $\hat{a}_t$ and the policy π, while the second term constrains the change in behavior by regularizing the difference between the original prompt z and the adapted prompt $L(z; \xi)$.

The adaptor layer L is initialized as an identity function at the start of the improvement stage. Additionally, during this stage, the actions in the offline dataset are relabeled as $\hat{a}_t$ to replace the original actions with improved ones, providing a new supervised target for prompt tuning.

4.4 Performance and Limitations

Generalization Ability Multi-Game Decision Transformer (MGDT), Prompt Decision Transformer (PromptDT), and Contextual Meta Transformer (CMT) demonstrate remarkable advancements in meta-RL with superior performance and generalization capabilities across diverse settings. MGDT excels in multi-game scenarios, achieving state-of-the-art results through pretraining with the Decision Transformer objective, robust scaling, and rapid fine-tuning with minimal data [7]. PromptDT showcases exceptional few-shot policy generalization by adapting to unseen tasks using short trajectory prompts, surpassing meta offline RL baselines like MACAW, and effectively handling out-of-distribution (OOD) tasks [14]. Similarly, CMT achieves impressive results across offline RL settings, including single-agent, meta-RL, and multi-agent tasks, often outperforming baselines like DT, BRAC, and CQL, while relying solely on zero-shot adaptation during evaluation [7].

Dataset Quality Dependence The performance of PromptDT and CMT heavily relies on high-quality expert data, as both models associate state-action sequences with desired outcomes based on demonstrations. PromptDT, using trajectory prompts, excels in few-shot generalization when provided with expert prompts but suffers significantly with lower-quality datasets [14]. Similarly, CMT leverages context trajectories during training, with expert trajectories yielding substantial performance gains, reinforcing its dependence on quality data [15]. In contrast, MGDT exhibits greater robustness, effectively learning from both expert and non-expert data. By employing an expert action inference mechanism inspired by discriminator-guided generation, MGDT filters valuable information from diverse datasets and outperforms models trained exclusively on expert data [7]. This distinction highlights the varying dependencies on dataset quality, with MGDT offering more flexibility and applicability in scenarios with limited optimal demonstrations.

5 Learning from Learning Histories: In-Context Reinforcement Learning

5.1 Algorithm Distillation

Most of the prior works that adapt transformer, specifically Decision Transformer, relies on expert trajectories data. It is more favorable to train those models on trajectories from a stable and optimal policy. Such a paradigm is limited by the scarcity of large-scale optimal trajectory data available. Algorithm Distillation (AD) was developed to address this limitation by enabling Transformers to learn from the learning process of an RL algorithm, even if that algorithm doesn't produce fully optimal trajectories [16].

Algorithm 5 AD Training and Evaluation

```
Input: Tasks $\mathcal{M}^{train} \cup \mathcal{M}^{test}$, Source RL algorithm $P_{\phi_i}$ for $i = 1, \cdots N$
procedure DATASET GENERATION($\mathcal{M}^{train}$)
    for $i = 1, \cdots, N$ do
        Sample a task $\mathcal{M}_i^{train} \sim \mathcal{M}^{train}$
        Train a source RL algorithm $P_{\phi_i}$ until converges to optimal policy
        Save learning history $h_T^{(i)} = \{(o_t, a_t, r_t)_{t=1}^T\}_i$ to dataset $\mathcal{D} \leftarrow \mathcal{D} \cup h_T^{(i)}$

procedure TRAINING($\mathcal{M}^{train}$)
    while $P_\theta$ not converged do
        Sample multi-episodic subsequence $\bar{h}_j^{(i)} = (o_t, a_t, r_t)_{t=j}^{j+c}$ of length $c$
        $P_\theta \leftarrow$ Transformer($\bar{h}_t^{(i)}|\theta$)    $\forall t = j, \cdots j + c, i = 1, \cdots N$
        $\mathcal{L}(\theta) = -\sum_{n=1}^{N} \sum_{t=1}^{T-1} \log P_\theta(a_t^{(n)}|h_{t-1}^{(n)}, o_t^{(n)})$
        $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$

procedure EVALUATION($\mathcal{M}^{test}$)
    Sample a task $\mathcal{M}_i^{test} \sim \mathcal{M}^{test}$
    Initialize context queue $C = \{\}$
    Unroll the policy based on transformer $P_\theta(.|C)$
    Calculate the return accumulated for each episode
```

The key insight is that even suboptimal trajectories contain valuable information about how an RL algorithm adapts and improves its policy over time. AD leverages this information by training a Transformer model on the learning histories of a source RL algorithm, which can include both expert and non-expert data.

AD is motivated by the assumption that agent's action is a function of its past experience. AD model is trained with dataset $\mathcal{D}$ consisted a a learning history generated by source algorithm P_{source} on many individual tasks $\{\mathcal{M}_n\}_{n=1}^{N}$.

$$\mathcal{D} = \{(o_1^{(n)}, a_1^{(n)}, r_1^{(n)}, \cdots, o_T^{(n)}, a_T^{(n)}, r_T^{(n)} \sim P_{\mathcal{M}_n}^{source}\}_{n=1}^{N}$$

Using a transformer model P_θ with parameter θ, the training objective is to minimize

$$\mathcal{L}(\theta) = -\sum_{n=1}^{N}\sum_{t=1}^{T-1} \log P_\theta(a_t^{(n)}|h_{t-1}^{(n)}, o_t^{(n)})$$

5.2 Headless Algorithm Distillation

Algorithm 6 Headless-AD Training and Evaluation

```
Input: State-action-reward trajectories $\{s_t, a_t, r_t\}$, Action set $A$
procedure TRAINING PHASE
    Initialize random embeddings for all actions $A$: $\{a_{emb,0}, \cdots, a_{emb,N}\}$
    while model not converged do
        Convert actions in the context $a^t$ to embeddings $a_{emb}^t$
        Predict the next action embedding $P_\theta(\hat{a}_{emb}^t|)$
        Compute similarity between $\hat{a}_{emb,t}$ and all $a_{emb,0}, \cdots, a_{emb,N}$ in $A$
        Generate action distribution $P_\theta(a_t|h_{t-1}, s_t, A)$
        Compute contrastive loss $\mathcal{L}(\theta)$ based on correct $a_t$
        $\theta \leftarrow \theta - \alpha\nabla_\theta\mathcal{L}(\theta)$

procedure EVALUATION PHASE
    Initialize context history $h_0 = \{\}$
    for $t < T$ do
        Predict action embedding $\hat{a}_{emb,t} \sim P_\theta(.|h_{t-1}, s_t, A)$
        Select action $a_t$ based on closest match with $\{a_{emb,0}, \cdots, a_{emb,N}\}$
        Execute action $a_t$ in the environment and observe $r_t, s_{t+1}$
        Update context $h_t \leftarrow h_{t-1} \cup \{s_t, a_t, r_t\}$
    Calculate accumulated return for the trajectory
```

The original Algorithm Distillation has a limitation due to its fixed action space setup. In addition, AD's performance also diminishes as the action semantics is changed (e.g. the dimension is permutated). Headless Algorithm Distillation (Headless-AD) addresses this problem by omitting the final layer and incorporate several modifications.

First, headless-AD removes the final linear layer, allowing the model to directly predict action embeddings instead of action probabilities. This breaks the dependence on the specific action space used during training. In addition, instead of fixed action embeddings, Headless-AD generates random embeddings for each action at the start of every training step. This forces the model to learn more general action representations based on contextual relationships [17].

To provide the model with information about the available actions in the current context, Headless-AD prepends the input sequence with an action embedding prompt that enumerates the embeddings for all possible actions. Finally, Headless-AD employs a contrastive loss function to train the model, encouraging it to predict action embeddings that are similar to the correct action embedding and dissimilar to those of incorrect actions.

Besides those modification, Headless-AD employs the same training and evaluation method as that of AD. The training and evaluation procedure is highlighted in Algorithm 6.

5.3 Evaluation and Limitations

Source Algorithm Flexibility and Generalization Based on the result, AD approach offers several advantages. First, it reduces the dependence on large, expert-only datasets as AD can successfully learn a policy improvement operator by imitating the learning histories of a source RL algorithm, even when those histories include non-expert data. Second, AD model can continue to adapt and improve its performance in-context as it interacts with a new environment, allowing it to adapt to new tasks without updating its network parameters. Finally, AD offers flexibility in the choice of the source RL algorithm. AD can distill various RL algorithms, allowing for customization based on specific learning behaviors or leveraging the strengths of different approaches [16].

Dynamic Action Space Even though AD is not robust to changes of action semantics, with a slight refinement from, Headless-AD effectively overcomes the action space limitations, demonstrating strong generalization to new action spaces of varying sizes, contents, and orderings. It achieves performance comparable to the original data generation algorithm across diverse environments, such as Bernoulli and contextual bandits, as well as gridworld scenarios. However, its capacity is still limited by the dimensionality of the action embeddings, restricting the number of actions it can handle [17].

6 Discussions

Complexity and Computing Resources Transformer-based reinforcement learning (RL) models face significant challenges with time complexity and computational demands, particularly due to the quadratic cost of self-attention. This slows down inference, making Transformers less suitable for tasks requiring rapid decision-making.

Context Limitation Similar to large language models (LLMs), transformer-based RL models face context limitations, hindering performance in long-episode tasks. While transformers improve memory, they offer little benefit for credit assignment in long-term tasks [18], highlighting their shortcomings in addressing longer-episode RL problems.

Data Scarcity Data scarcity poses a significant challenge for offline reinforcement learning (RL) methods. As research increasingly shifts towards developing more generalized agents capable of operating across multiple domains and environments, the need for large-scale datasets becomes critical. Such datasets are essential for pretraining RL models using trajectory optimization approaches, including expert and optimal trajectories for behavior cloning setups, as well as diverse learning histories for Algorithm Distillation (AD)-based methods. However, acquiring these large-scale datasets presents considerable difficulties, as it often requires extensive resources, access to high-quality expert demonstrations, and comprehensive coverage of diverse environments and tasks.

7 Conclusion

Trajectory optimization reframes reinforcement learning (RL) as a sequence modeling problem. It began with behavioral cloning, simplifying RL to conditioned next-action prediction, as seen in methods like Decision Transformer (DT) and Trajectory Transformer (TT). Newer approaches, such as Multi-Game DT, PromptDT, and CMT, introduced task generalization by leveraging expert reward prediction or prompts to guide action predictors. The latest advancement, Algorithm Distillation, trains models using learning histories, improving adaptability to new tasks while reducing dependence on expert data. Despite the performance improvement, transformer-based RL remains constrained by complexity, limited context handling, and data scarcity. Future research should prioritize resource-efficient transformer models, generalized RL for multi-domain tasks with dynamic action and state spaces, and advancements in contextual and in-context reinforcement learning through trajectory optimization.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018.

[2] R. S. Sutton, "Temporal credit assignment in reinforcement learning," AAI8410337, Ph.D. dissertation, 1984.

[3] E. Korkmaz, *A survey analyzing generalization in deep reinforcement learning*, 2024. arXiv: 2401.02349 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2401.02349.

[4] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1706.03762.

[5] M. Janner, Q. Li, and S. Levine, *Offline reinforcement learning as one big sequence modeling problem*, 2021. arXiv: 2106.02039 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2106.02039.

[6] L. Chen, K. Lu, A. Rajeswaran, *et al.*, *Decision transformer: Reinforcement learning via sequence modeling*, 2021. arXiv: 2106.01345 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2106.01345.

[7] K.-H. Lee, O. Nachum, M. Yang, *et al.*, *Multi-game decision transformers*, 2022. arXiv: 2205.15241 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2205.15241.

[8] Z. Xie, Z. Lin, D. Ye, *et al.*, *Future-conditioned unsupervised pretraining for decision transformer*, 2023. arXiv: 2305.16683 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2305.16683.

[9] Y. Sun, S. Ma, R. Madaan, *et al.*, *Smart: Self-supervised multi-task pretraining with control transformers*, 2023. arXiv: 2301.09816 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2301.09816.

[10] X. Liu, X. Lou, J. Jiao, and J. Zhang, *Position: Foundation agents as the paradigm shift for decision making*, 2024. arXiv: 2405.17009 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2405.17009.

[11] M. Phuong and M. Hutter, *Formal algorithms for transformers*, 2022. arXiv: 2207.09238 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2207.09238.

[12] S. Hu, L. Shen, Y. Zhang, Y. Chen, and D. Tao, *On transforming reinforcement learning by transformer: The development trajectory*, 2023. arXiv: 2212.14164 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2212.14164.

[13] T. Yamagata, A. Khalil, and R. Santos-Rodriguez, *Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline rl*, 2023. arXiv: 2209.03993 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2209.03993.

[14] M. Xu, Y. Shen, S. Zhang, *et al.*, *Prompting decision transformer for few-shot policy generalization*, 2022. arXiv: 2206.13499 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2206.13499.

[15] R. Lin, Y. Li, X. Feng, *et al.*, *Contextual transformer for offline meta reinforcement learning*, 2022. arXiv: 2211.08016 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2211.08016.

[16] M. Laskin, L. Wang, J. Oh, *et al.*, *In-context reinforcement learning with algorithm distillation*, 2022. arXiv: 2210.14215 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2210.14215.

[17] V. Sinii, A. Nikulin, V. Kurenkov, I. Zisman, and S. Kolesnikov, *In-context reinforcement learning for variable action spaces*, 2024. arXiv: 2312.13327 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2312.13327.

[18] T. Ni, M. Ma, B. Eysenbach, and P.-L. Bacon, *When do transformers shine in rl? decoupling memory from credit assignment*, 2023. arXiv: 2307.03864 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2307.03864.