STARFORMER: TRANSFORMER WITH STATE-ACTION-REWARD REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement Learning (RL) can be considered as a sequence modeling task, i.e., given a sequence of past state-action-reward experiences, a model autoregressively predicts a sequence of future actions. Recently, Transformers have been successfully adopted to model this problem. In this work, we propose **St**ate-**A**ction-**R**eward Transformer (**StAR**former), which explicitly models strongly-related local causal relations to help improve action prediction in long sequences. StARformer first extracts local representations (i.e., StAR-representations) from each group of state-action-reward tokens within a very short time span. A sequence of such local representations combined with state representations, is then used to make action predictions over a long time span. Our experiments show that StARformer outperforms the state-of-the-art Transformer-based method on Atari (image) and Gym (state vector) benchmarks, in both offline-RL and imitation learning settings. StARformer is also more compliant with longer sequences of inputs compared to the baseline. The code will be released online.

1 INTRODUCTION

Reinforcement Learning (RL) naturally comes with sequential data: an agent observes a state from the environment, takes an action, observes the next state and receives a reward from the environment. With the recent advances, RL has been formulated as a sequential decision-making task, and Transformer (Vaswani et al., 2017) architectures have been adopted as generative trajectory models to solve it, i.e., given past experiences of an agent composed of a sequence of state-action-reward triplets, a model iteratively generates an output sequence of action predictions (Chen et al., 2021; Janner et al., 2021). This newly introduced formulation has shown to be useful, especially in terms of its capability to model long-term sequences (Janner et al., 2021) and sequence distributions (Chen et al., 2021).

In the state-of-the-art Transformer models for RL such as Chen et al. (2021); Janner et al. (2021), an input sequence is plainly processed through self-attention – the core modeling component of Transformers (Vaswani et al., 2017). In fact, here, states, actions, and rewards are considered as individual tokens (unit representations), and self-attention is applied to model their pairwise relations. This way, a given state, action, or reward token may attend to any of the (previous) tokens in the sequence, and this allows the model to capture long-term relations.

However, states, actions, and rewards within closed timesteps are strongly connected by significant causal relations. For instance, the states in the recent past have a stronger effect on the next action, compared to the states in distant past. Similarly, the immediate-future state and the corresponding reward are direct results of the current action. In such a setting, a Transformer attending to all tokens naively may suffer from excess information and dilute the truly-essential relation priors, especially when inputs are quite long, either in spatial (Yang et al., 2021) or temporal (Janner et al., 2021) dimension, or with the Transformer layers going deep (Touvron et al., 2021b). Therefore, we believe that learning a short-term representation from these strongly-related elements (i.e., tokens in this case) explicitly, can benefit from strong relation priors and eventually help long-term sequence modeling in RL.

To this end, we propose **St**ate-**A**ction-**R**eward Transformer (**StAR**former), a Transformer architecture learning **St**ate-**A**ction-**R**eward-representations (i.e., **StAR**-representations) for sequence modeling in RL. StARformer comes with two basic components: a Step Transformer and a Sequence Transformer. The Step Transformer learns local representations (i.e., StAR-representations) explicitly, based on strongly-related state-action-reward tokens within a timestep. The Sequence Transformer then combines StAR-representations and state representations over a long-term span to make a sequence of action predictions. In our experiments, we show that StARformer outperforms the state-of-the-art Transformer-based method in both offline-RL and imitation learning settings, while being more compliant with longer input sequences in comparison.

2 Preliminary

2.1 Transformer

Transformer (Vaswani et al., 2017) architectures showed a diverse application in language (Devlin et al., 2018) and vision tasks (Dosovitskiy et al., 2020; Arnab et al., 2021). Given a sequence of input tokens $X = \{x_i\}, X \in \mathbb{R}^{n \times d}$, a Transformer layer maps it to an output sequence of tokens $Z = \{z_i\}, Z \in \mathbb{R}^{n \times d}$ through Multi-headed Self-Attention (MSA) (Devlin et al., 2018), followed by Multi-Layer Perceptron blocks (MLP) with residual connection (He et al., 2016) and Layer Normalization (LN) (Ba et al., 2016): $Z' = \text{MSA}(\text{LN}(X)) + X, Z = \text{MLP}(\text{LN}(Z')) + Z'$. A Transformer model is obtained by stacking multiple such layers. We denote the mapping for each layer (l) as $F(\cdot)$: $Z^l = F(Z^{l-1})$. We use the notation $F(\cdot)$ to represent a Transformer layer in the remaining sections.

Self-attention (Lin et al., 2017; Parikh et al., 2016; Cheng et al., 2016; Vaswani et al., 2017) is the core component of Transformer, which models pairwise relations. In Vaswani et al. (2017), an input token representation X is linearly mapped into query, key and value representations, i.e., $\{Q, K, V\} \in \mathbb{R}^{n \times d}$, to compute self-attention as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d}})V.$$

The idea is to aggregate values based on pairwise similarity computed from queries and keys. In such mechanism, each token can "attend to", i.e. aggregate all tokens in the sequence, with a specified weight.

Vision Transformer (ViT) (Dosovitskiy et al., 2020) extends the same idea of self-attention to the image domain, by tokenizing input images, i.e., extracting a set of non-overlapping image patches to create a sequence of tokens. Given an input image $s \in \mathbb{R}^{H \times W \times C}$, a set of n non-overlapping local patches $P = \{p_i\} \in \mathbb{R}^{h \times w \times C}$ is extracted, flattened and linearly mapped to a sequence of tokens $\{x_i\} \in \mathbb{R}^d$, through a fully-connected (FC) layer $\mathbb{R}^{hwc} \rightarrow \mathbb{R}^d$. We use a similar tokenizing approach in ViT for a part of our state token embeddings.

2.2 RL as Sequence Modeling

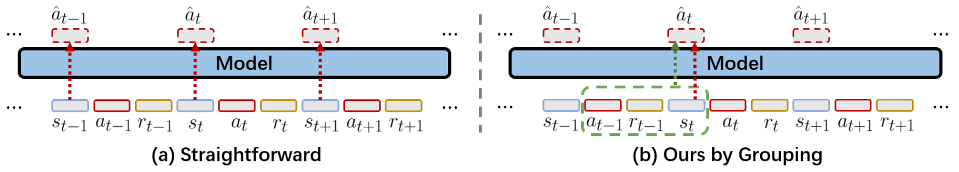

Figure 1: Illustration of RL as sequence modeling (using Transformer). (a) A straightforward approach. (b) Our improvement. The concept is local features helps the long-term modeling. Red arrows stand for action is "autoregressively generated" by state. Green arrow stands for this local representation contributes the action generation together with state representation.

We consider a Markov Decision Process (MDP), described by tuple $(\mathcal{S}, \mathcal{A}, P, \mathcal{R})$, where $s \in \mathcal{S}$ represents the state, $a \in \mathcal{A}$, the action, $r \in \mathcal{R}$, the reward, and P, the transition dynamics given by $P(s'|s, a)$. In MDP, a trajectory (τ) is defined as the past experience of an agent, which is a sequence composed of states, actions, and rewards in the following temporal order $\tau = \{r_1,\ s_1,\ a_1,\ r_2,\ s_2,\ a_2,\ \ldots,\ r_t,\ s_t,\ a_t\}$. RL as a sequence modeling task is formulated as making action predictions from past experience (Chen et al., 2021; Janner et al., 2021) according to:

$$Pr(\hat{a}_t) = p(a_t|\ r_{1:t},\ s_{1:t},\ a_{1:t-1}). \tag{1}$$

Recent work (Chen et al., 2021; Janner et al., 2021) try to adopt an existing Transformer architecture (Radford et al., 2019) to RL in this formulation. In Chen et al. (2021); Janner et al. (2021), states (s), actions (a), and rewards (r) are considered as input tokens (see Figure 1a), while using a

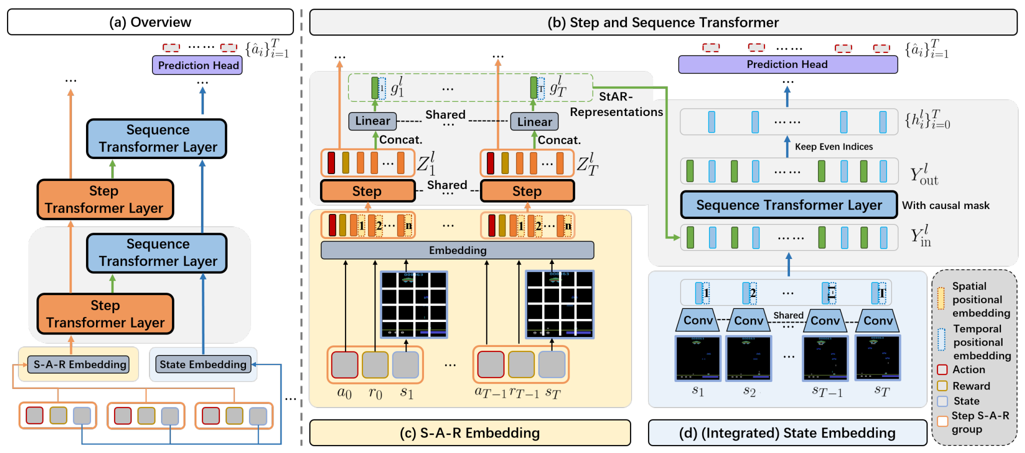

Figure 2: Architecture of the proposed StARformer model (with an image-based state representation). (a) Network overview, which shows how, Step Transformer and Sequence Transformer are connected at each layer. (b) Detailed functionality of each component. Step Transformer processes T separate groups of tokens simultaneously with shared weights. We learn StAR-representation g_t^l by aggregating output tokens from Step Transformer and use it in Sequence Transformer. We only keep tokens with even indices and discard the others in Sequence Transformer's output since we generate actions from states. (c) A trajectory is segmented into local groups of (a_{t-1}, r_{t-1}, s_t) and embeded as tokens. (d) States are separately embedded using convolutions with shared weights.

causal mask to ensure an autoregressive output sequence generation (i.e. following Eq. 1). Here, a token can only access its corresponding previous tokens—in time— through self-attention.

In contrast, our formulation models strong causal relations explicitly, while attending to long-term relations separately. To do this, in this work, we break a trajectory into small groups of state-action-reward tuples (i.e., s, a, r), and learn explicit local relations within the tokens of each group through self-attention (see Figure 1b). We further model long-term relations together with learned local relations inclusively. Our grouping is based on the intuition that the local causal relations between s, a and r are strong, i.e., the reward r_{t-1} and the state s_t are direct results of the action a_{t-1}. Therefore, we propose our StARformer to learn strong intermediate representations from local groups of $(a_{t-1},\ r_{t-1},\ s_t)$ explicitly, to help improve long-term sequence modeling.

3 Method

3.1 Overview

StARformer consists of two basic components: Step Transformer and Sequence Transformer, together with interleaving connections (see Figure 2a). Step Transformer learns StAR-representations from strongly-connected local tokens explicitly, which are then fed into the Sequence Transformer along with integrated state representations to model a long-term sequence of trajectory. At the output of the final Sequence Transformer layer, we make action predictions via a prediction head. In the following subsections, we introduce the two Transformer components in detail.

3.2 Step Transformer

3.2.1 State-Action-Reward Embeddings

Grouping state-action-reward Our intuition is to model strong local relations explicitly in the Step Transformer. To do so, we first segment a trajectory (τ) into a set of groups, where each group consists of previous action (a_{t-1}), reward (r_{t-1}) and current state (s_t)[1]. See Figure 2b. Each element within a group has a strong causal relation with the others.

Patch-wise state token embeddings To create a more descriptive state representation, we tokenize each input state by dividing it into a set of non-overlapping "spatial" patches along the (part of) their dimensions. We can consider such patches for both image or vector input states, i.e., the two main types of input state representations in RL. In the case of image states $s_t \in \mathbb{R}^{H\times W\times C}$, we

[1]We pad the trajectory with a null action and zero reward at the initial state s_1 of the trajectory (see Section 3.4)

divide each image into patches $\{s_t^i\}, \in \mathbb{R}^{h \times w \times C}$, similar to ViT (Dosovitskiy et al., 2020), which gives HW/hw state tokens in total per image. In the case of vector states $s_t \in \mathbb{R}^C$, we simply consider each channel s_t^i as a "spatial" patch from its channel space, and obtain C state tokens in total per vector. We use a linear projection— weights of which are shared by each state patch across all groups— to create token embeddings ($\mathbb{R}^{hwC} \rightarrow \mathbb{R}^d$ for image states or $\mathbb{R}^1 \rightarrow \mathbb{R}^d$ for vector states):

$$z_{s_t^i} = \begin{cases} \text{FC}(\text{Flatten}(s_t^i)) + e_i^{\text{spatial}} & \text{for image states,} \\ \text{FC}(s_t^i) + e_i^{\text{spatial}} & \text{for vector states,} \end{cases}$$

where $e_i^{\text{spatial}} \in \mathbb{R}^d$ represents the spatial positional embedding for each patch location, n, the number of patches and d, the embedding dimension. Note that there are no temporal positional embeddings for patch-wise spatial tokens since they are processed agnostic of timesteps.

Our motivation in creating a set of patch-wise tokens is to allow the Step Transformer to model the relations of actions and rewards with state localities. We believe that it encodes more (spatially) descriptive local information in the learned StAR-representation, and eventually helps the task of sequential action prediction.

Action and reward token embeddings We simply embed the action tokens with a linear projection, and the reward tokens with a linear projection followed by a Tanh$(\cdot)$ activation function similar to Chen et al. (2021) to smooth a proper value range:

$$\begin{aligned} z_{a_{t-1}} &= \text{FC}(a_{t-1}), \\ z_{r_t} &= \text{Tanh}(\text{FC}(r_t)). \end{aligned}$$

Altogether, we get a state-action-reward representation as the input to the initial Step Transformer layer which is given by: $Z_t^0 = \{z_{a_{t-1}},\ z_{r_t},\ z_{s_t^1},\ z_{s_t^2},\ \ldots,\ z_{s_t^n}\}$. We have T groups of such token representations per trajectory, which are simultaneously processed by the Step Transformer with shared parameters.

3.2.2 Step Transformer Layer

We adopt the conventional Transformer design from Vaswani et al. (2017) (see Section 2.1) as our Step Transformer layer. Each group of tokens from the previous layer Z_t^{l-1} is transformed to Z_t^l by a Step Transformer layer with the mapping F_{step}^l: $Z_t^l = F_{\text{step}}^l(Z_t^{l-1})$.

At the output of each Step Transformer layer l, we further obtain a State-Action-Reward-representation (StAR-representation) $g_t^l \in \mathbb{R}^D$ by aggregating output tokens $Z_t^l \in \mathbb{R}^{n \times d}$ (see green flows in Figure 2b). We do this by concatenating tokens of each group Z_t^l and linearly projecting ($\mathbb{R}^{nd} \rightarrow \mathbb{R}^D$), where d and D correspond to embedding dimensions of Step Transformer and Sequence Transformer respectively:

$$g_t^l = \text{FC}([Z_t^l]) + e_t^{\text{temporal}}.$$

Here $[\cdot]$ represents concatenation of the tokens within each group and $e_t^{\text{temporal}} \in \mathbb{R}^D$, the temporal positional embeddings for each timestep. It is important to note that we add such temporal positional embeddings to g_t^l at each layer, as $[Z_t^l]$ is learned agnostic in time. Finally, The output StAR-representation g_t^l is fed into the corresponding Sequence Transformer layer for long-term sequence modeling.

3.3 Sequence Transformer

Our Sequence Transformer models long-term sequences by looking at StAR-representations and the *integrated state tokens* (introduced below) over the whole trajectory.

3.3.1 Integrated state token embeddings

In contrast to the patch-wise token embeddings in Step Transformer, we embed the input states s_t (image or vector) as a whole, to create integrated state tokens h_t^0 to be fed into the Sequence Transformers. Each such token represents a single state representation, describing the state globally in space. We do this by processing each state through multiple convolutional layers (in the case of image states), or through multiple fully-connected layers (in the case of vector states). We use the convolutional encoder from Mnih et al. (2015).

$$h_t^0 = \begin{cases} \text{Conv}(s_t) + e_t^{\text{temporal}} & \text{for image states,} \\ \text{FC}(s_t) + e_t^{\text{temporal}} & \text{for vector states,} \end{cases}$$

where $e_t^{\text{temporal}} \in \mathbb{R}^D$ represents the temporal positional embeddings exactly the same as we add to g_t for each timestep. This difference in state embeddings (patch-wise vs integrated) is motivated by the potential benefits of the representation diversity, which is empirically validated in our ablations in Section 5.4.

3.3.2 Sequence Transformer Layer

Similar to Step Transformer, we use the conventional Transformer layer design from Vaswani et al. (2017) for our Sequence Transformer. The input to the Sequence Transformer layer l consists of representations from two sources: (1) the learned StAR-representations $g_t^l \in \mathbb{R}^D$ from the corresponding Step Transformer layer, and (2) the integrated state representation $h_t^{l-1} \in \mathbb{R}^D$ from the previous Sequence Transformer layer. The two types of token representations are merged to form a single sequence, preserving their temporal order (as elaborated below):

$$Y_{\text{in}}^l = \{g_1^l,\ h_1^{l-1},\ g_2^l,\ h_2^{l-1},\ \ldots,\ g_T^l,\ h_T^{l-1}\}. \tag{2}$$

We place g_t^l before h_t^{l-1} —which originates from s_t— because g_t^l contains information of the *previous* action a_{t-1}, which comes prior to s_t in the trajectory. We also apply a causal mask in the Sequence Transformer to ensure that the tokens at time t cannot attend any future tokens (i.e., $> t$).

Sequence Transformer computes an intermediate set of output tokens $Y_{\text{out}}^l = F_{\text{sequence}}^l(Y_{\text{in}}^l)$. We then select the tokens at even indices of Y_{out}^l (where indexing starts from 1) to be the integrated state tokens h_t^l, which are fed into the next Sequence Transformer layer. We do this because the even indices correspond to the tokens originated from the integrated state representations s_t (see Figure 2c top right), and should be used to predict actions (see Figure 1) from an autoregressive perspective. Tokens in Y_{out}^l with odd indices are discarded.

$$\begin{aligned} Y_{\text{out}}^l &= \{y_{\text{out};1}^l,\ y_{\text{out};2}^l,\ \ldots,\ y_{\text{out};2T}^l\}, \\ h_i^l &:= y_{\text{out};2i}^l. \end{aligned}$$

Therefore, the overall input (StAR-representation and integrated state representation) to the subsequent layer can be given as (by rewriting Eq. 2):

$$Y_{\text{in}}^{l+1} = \{g_1^{l+1},\ h_1^l,\ g_2^{l+1},\ h_2^l,\ \ldots,\ g_T^{l+1},\ h_T^l\} = \{g_i^{l+1},\ y_{\text{out};2i}^l\}_{i=1}^T$$

3.3.3 Action Prediction

The output of the last Sequence Transformer layer (after selection as mentioned above) is used to make action predictions by processing it through a prediction head $\phi(\cdot)$ linearly mapping to the action dimension (with shared weights for all timesteps): $\hat{a}_t = \phi(h_t^l)$ (see Appendix for more details).

3.4 Training and Inference

During training, we provide trajectories τ with a length T, randomly sampled (and sliced) from a memory buffer. We use cross-entropy loss for a discrete action space and the mean-squared error (MSE) for a continuous action space. The overall loss term for a given training sequence is the loss averaged across all T predictions.

At inference, we initialize an input trajectory as $\tau = \{a_0,\ r_0,\ s_1\}$, where a_0 is a null action[2] and $r_0 = 0$ is a zero reward which we pad at the start of each trajectory. StARformer makes an initial prediction $\hat{a}_1$ based on τ and receive next state s_2 and reward r_2. We concatenate these $\hat{a}_1$, s_2 and r_2 to the trajectory τ and make iterative predictions until one episodes ends.

[2]See Appendix for details on defining null actions.

4 Experiment Settings

Settings We use offline RL (Levine et al., 2020) and imitation learning as our experiment settings. These are commonly used in related work which formulates RL as a sequence modeling task (Chen et al., 2021; Janner et al., 2021), because ground truth labels can be obtained for actions to train a sequence model. In offline RL, we have a fixed memory buffer of sub-optimal trajectory rollouts. Offline RL is generally more challenging compared to conventional RL (Levine et al., 2020) due to the shifted distribution.

In imitation learning, however, an agent is not exposed to reward signals. Therefore, in this setting, we simply keep the same sub-optimal dataset from offline RL and remove the rewards when creating the trajectory sequences. This is an even harder problem compared to traditional imitation learning, where the provided trajectories are usually expert demonstrations (i.e. with high rewards). The only difference in terms of the model structure is that we now have one fewer reward token at the input of Step Transformerin our model, and T fewer reward tokens in the baseline model (T is the number of time-steps in the input trajectory).

Environments We consider Atari (Bellemare et al., 2013) (image state, discrete action space) and OpenAI Gym (Brockman et al., 2016) (vector state, continuous action space) to evaluate our model in different input modalities and action spaces. We pick 6 games in Atari: Assault, Boxing, Breakout, Pong, Qbert, and Seaquest. We select 3 environments in OpenAI Gym (Brockman et al., 2016): Hopper, Walker, and HalfCheetah. The same environments are used in Chen et al. (2021) as well. We use the same offline RL datasets as in Chen et al. (2021) to perform a thorough and fair comparison. In fact, we use the Atari dataset from Agarwal et al. (2020) and the Gym dataset from Fu et al. (2020).

We report the absolute value of average episode return (i.e., cumulative reward) from evaluation rollouts. Results with mean and standard deviation are from multiple runs across 3 random seeds.

5 Results

Table 1: Evaluation episode returns of our proposed StARformer (SF) and the baseline Decision-Transformer (DT) Chen et al. (2021) on Atari and Gym environments. The highest scores in each setting and environment are in bold. Gain is the relative improvement (percentage) of our method compared to the baseline. We also present the reward statistics of the training datasets for reference.

Setting	Method	Atari Environment					
		Assault	Boxing	Breakout	Pong	Qbert	Seaquest
offline RL	DT	504 ± 54	78.3 ± 4.6	70.7 ± 8.1	12.8 ± 3.2	3782 ± 695	**1007 ± 170**
	SF (ours)	**772 ± 215**	**80.7 ± 5.6**	**132.1 ± 21.5**	**16.4 ± 2.7**	**6584 ± 2643**	771 ± 366
	(Gain)	53.1%	3.1%	97.0%	28.2%	74.1%	-23.4%
Imitation	DT	595 ± 89	72 ± 2.6	54.3 ± 1.2	7.7 ± 2.1	2099 ± 1075	826 ± 118
	SF (ours)	**867 ± 201**	**79.4 ± 2.6**	**89.0 ± 13.0**	**15.4 ± 2.8**	**6448 ± 751**	**915 ± 121**
	(Gain)	45.6%	10.2%	63.8%	100.9%	207.2%	10.7%
Dataset (Max.)		153	98	92	21	600	290

Setting	Method	Gym Environment		
		HalfCheetah	Hopper	Walker2d
offline RL	DT	10799 ± 74.1	**3578 ± 13.8**	4995.3 ± 18.6
	SF (ours)	**11278 ± 20.7**	3549 ± 27.9	**5010.3 ± 49.9**
	(Gain)	4.4%	-0.8%	0.3%
Imitation	DT	10805 ± 130.3	3572 ± 4.0	5024.3 ± 26.5
	SF (ours)	**11315 ± 15.9**	**3592 ± 4.7**	**5034.3 ± 11.2**
	(Gain)	4.7%	0.6%	0.2%
Dataset (Mean ± Std.)		10656 ± 441.7	3511 ± 328.6	4920.5 ± 136.4

5.1 Can StARformer improve sequence modeling for RL?

We first compare our StARformer (SF) with the state-of-the-art Transformer-based RL method in Atari and Gym environments, under both offline RL and imitation learning settings. We select the Decision-Transformer proposed in Chen et al. (2021), (referred as DT) as our baseline. Here, we keep $T = 30$ for Atari and $T = 20$ for Gym, similar to (Chen et al., 2021), where T is the number of

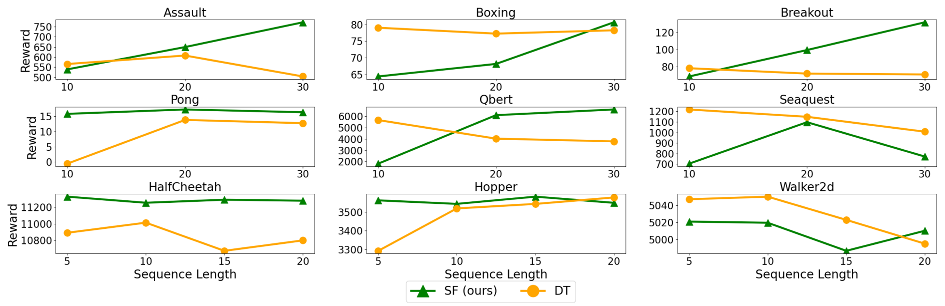

Figure 3: Performance under different trajectory lengths. $T \in \{10, 20, 30\}$ in Atari and T ∈{5,10,15,20} in Gym. In most of the cases, StARformer shows a better performance when increasing the trajectory length, and surpasses that of the baseline, validating that our method can effectively model long-term sequences.

time-steps (length) of each input trajectory (τ). Evaluation rollouts automatically end in Atari when the game is over, and after a fixed number of steps (1000 steps) in Gym.

Table 1 shows that our method outperforms the baseline almost always, in both offline RL and imitation learning settings. StARformer outperforms the baseline by a large margin in Atari, showing that our method can model image state inputs very well. Our method also has a lower standard deviation than the baseline in Gym under imitation learning while achieving higher rewards on average. This suggests our StARformer gains better stability in modeling trajectories without the guidance of reward signals.

We further compare the change in performance, when switching from offline RL to imitation, showing how reward signals affect the sequence modeling. In fact, Table 1 shows that in Atari, StARformer performs comparably (except for Breakout) in both settings, but the performance of the baseline has dropped in imitation learning setting. It validates that StARformer can model RL as a sequence prediction task, even without a reward signal, whereas the baseline considerably depends on the reward. However, in Gym environments, removing rewards surprisingly results in minor performance changes for both of the methods. We believe this is because Transformers are sensitive to the highly dense rewards in Gym as opposed to Atari.

5.2 Can StARformer model long trajectories?

Longer sequences are usually more challenging to model, but at the same time, they should benefit action predictions due to extra information. We vary the input sequence length T in both Atari and Gym environments and evaluate the effect on offline RL setting. The results are shown in Figure 3. It shows in general, StARformer performs better when increasing the trajectory length, often outperforming the baseline, which validates the claim that our method can model long-term sequences effectively.

5.3 Which reward setting: Step-wise reward or Reward-to-go?

Table 2: Results in step-wise reward and reward-to-go settings (under offline RL).

Method	Setting	Assault	Boxing	Breakout	Pong	Qbert	Seaquest
SF (ours)	<u>step-wise</u>	<u>772</u>	<u>80.7</u>	<u>132.1</u>	<u>16.4</u>	<u>6584</u>	<u>771</u>
	RTG	420	60.2	23.1	0.7	1748	670
DT	<u>RTG</u>	<u>504</u>	<u>78.3</u>	<u>70.7</u>	<u>12.8</u>	<u>3782</u>	<u>1007</u>
	step-wise	452	66.2	64.9	11.0	3494	611

We are also interested in how different reward settings work with sequence modeling in RL, and consider the effect of step-wise reward and reward-to-go (RTG) (Chen et al., 2021). Step-wise reward r_t is the raw reward generated by an environment for each step, which is usually considered in most of RL algorithms. Our StARformer also uses this step-wise reward. Decision-Transformer (Chen et al., 2021), however, originally uses RTG $\hat{R}_t$, which is the sum of future step-wise rewards: $\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$. We perform an experiment applying both the reward settings to each method, results of which are shown in Table 2. The original reward settings used by each method is underlined. We observe that none of the methods work well when the reward formulation has changed. StAR-representation even shows a larger performance drop under RTG. This is because our method benefits from modeling the information over a short-term span, which conflicts with RTG as it encodes the information from the future.

5.4 Ablations

We use the Atari environments to conduct our ablations because: a) the episode returns in Atari are more obvious and — in most cases — have no upper limit, b) Atari has images as input states, which is better for studying the effect of integrated state tokens — used by the Sequence Transformer — and, c) it is more suited to investigate the model capacity.

Integrated state tokens Here, we investigate the effect of integrated state tokens h_t^l in long-term modeling i.e., in Sequence Transformer, and how we encode such state tokens. In addition to our approach which uses convolutional layers, we implement two other variants: (a) We remove integrated state tokens, meaning that Sequence Transformer learns solely from StAR-representations $\{g_t^h\}$ (referred to as Ours w/o conv). (b) Similar to our patch-wise embedding, we encode state inputs to the Sequence Transformerusing another patch embedding (referred to as Ours (conv. $\rightarrow$ ViT)). It is worth noting that for the variant (b), the embedding modules in our two Transformers do not share the same parameters. Results of this ablation are given in Table 3. We observe that our method with

Table 3: Ablation results on state representations of the Sequence Transformer (imitation learning)

Method	Assault	Boxing	Breakout	Pong	Qbert	Seaquest
Ours (w/ conv.)	**867**	**79.4**	**89**	**15.4**	**6448**	915
Ours w/o conv.	580	78.7	35	14	4358	**936**
Ours (conv $\rightarrow$ ViT)	508	53	33	13	1825	536

convolutional integrated state tokens performs the best. Either removing or replacing the convolutional encoder results in a performance drop. We believe that the diversity between convolutional and ViT representations helps the model perform better— as also seen in model distillation (Touvron et al., 2021a) or model ensembles.

Step-to-Sequence layer-wise connection In StARformer, the Step Transformer is connected to the Sequence Transformer via g_t^l in a layer-wise manner (i.e., at each corresponding layer). We further investigate why our connection method is important by comparing with two other variants: (a) g_t^l is fused with h_t^l by summation (referred as Ours Fusion, see Figure 4(a)). (b) the only connection between the two is from the last layer of Step Transformer to the first layer of Sequence Transformer , in which, the Sequence Transformer is "stacked" on-top of the Step Transformer (referred as Ours Stack, see Figure 4(b)). Results are shown in Table 4. It shows that our per-layer connection method is important for long-term prediction, which is fusing information from StAR-representation in a layer-wise manner, and is done through attention rather than direct fusion.

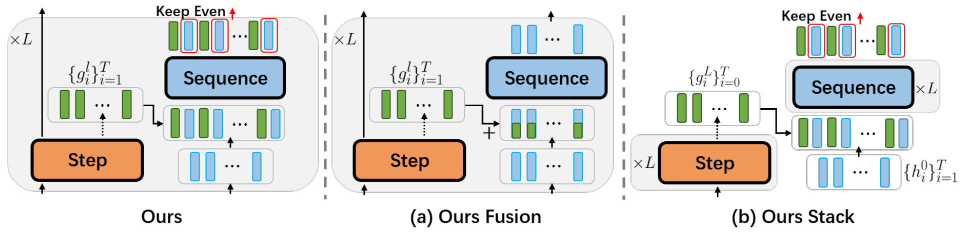

Figure 4: Illustration for our ablation study Transformer connection. We omit positional embeddings for simplicity. g^l (in green) is the StAR-representation from l-th layer of the Step Transformer and h^0 (in blue) is the initial integrated state token embeddings for Sequence Transformer. L is the number of layers.

Table 4: Ablation results on Transformer connectivity (offline RL)

Method	Assault	Boxing	Breakout	Pong	Qbert	Seaquest
Ours	772	**80.7**	**132.1**	**16.4**	**6584**	**771**
Ours Fusion	801	69.1	31.1	7.6	4248	606
Ours Stack	**939**	64.9	30.9	13.7	575	361

Does StARformer win by model capacity? Our model has $\sim$14M parameters, which is about 10 times larger than the baseline DT (Chen et al., 2021). Most of our parameters corresponds to patch-wise state embeddings. To see if the benefit of our method comes from extra parameters, we take an intuitive approach, replacing the baseline state encoder with our patch-wise state embedding (referred as DT with ViT). This makes the baseline and our method comparable in terms of parameters, results of which is shown in Table 5. We see that our model still outperforms the modified baseline

Table 5: Ablation results on model capacity (offline RL)

Method	Assault	Boxing	Breakout	Pong	Qbert	Seaquest
Ours	**772**	**80.7**	**132.1**	**16.4**	**6584**	771
DT with ViT	608	74.3	47.0	2.7	1135	885
DT	504	78.3	70.7	12.8	3782	**1007**

in 5 out of 6 games with a comparable number of parameters. It validates that simply increasing the model capacity may not improve the performance or even degrade in some cases. In fact, the modeling mechanism of our method is how it outperforms the baseline.

6 Related Work

6.1 Reinforcement Learning

Reinforcement Learning (RL) is usually modeled as a Markov Decision Process (MDP). In this setting, an action is made solely based on the current state according to the Markov property. Based on Markov property, single-step value-estimation methods have been derived from the Bellman equation, including Q-learning (Watkins & Dayan, 1992) and Temporal Difference (TD) learning, along with many other variants such as SARSA (Rummery & Niranjan, 1994), TD(λ) (Sutton, 1988), TD-Gammon (Tesauro et al., 1995), and Actor-Critic (Konda & Tsitsiklis, 2000). In recent work, neural networks are used to approximate the value function in value-based methods, introducing Deep Q-learning (Mnih et al., 2013).

Most recent directions (Chen et al., 2021; Janner et al., 2021) formulate RL as a sequence modeling task, i.e., given a sequence of recent experiences including state-actions-reward triplets, a model predicts a sequence of actions. This can potentially replace value-estimation methods and can be trained as supervised learning. However, this needs pre-existing trajectories (collected in advance), which makes it more compliant with offline RL and imitation learning settings.

6.2 Transformers

Transformer architectures (Vaswani et al., 2017) have been first introduced in language processing tasks (Devlin et al., 2018; Radford et al., 2018; 2019), to model interactions between a sequence of unit representations (i.e., *tokens*), in this case, word embeddings. Recently, however, Transformers have been adopted in vision tasks with the key idea of breaking down images/videos into tokens (Dosovitskiy et al., 2020; Arnab et al., 2021; Chen et al., 2020; Neimark et al., 2021), often outperforming convolutional networks (CNNs) in practice. Chen et al. (2021) exploit how GPT (Radford et al., 2019) can be applied to RL under sequence modeling settings.

Sequence modeling in RL is similar to learning from videos in terms of input data, which are composed of sequences of images (states). One challenge of applying Transformers on videos is the large number of input tokens. This problem has been explored in multiple directions, including attention approximation (Choromanski et al., 2020; Wang et al., 2020; Kitaev et al., 2019), separable attention in different dimensions (Arnab et al., 2021; Bertasius et al., 2021), reducing the number of tokens using local windows (Liu et al., 2021a;b) or adaptively generating a small amount of tokens (Ryoo et al., 2021).

StARformer shares a similar concept of using local windows for self-attention as Liu et al. (2021a). Our approach is also closely related to "divided space-time attention" in Bertasius et al. (2021) and "factorized self-attention" in Arnab et al. (2021), in terms of performing spatial followed by temporal attention. Our Step-to-Sequence connection is inspired by (Han et al., 2021), which is designed for image domain. However, our model is still unique as (a) we perform "spatial" (using Step Transformer) and "temporal" (using Sequence Transformer) attention in multiple Transformer layers, (b) we use separate sets of tokens having different origins, and (c) We feed a set of locally-attended tokens to perform self-attention with a set of global tokens for long-term sequence modeling.

7 Conclusion

In this work, we introduce StARformer, which models strongly connected local relations explicitly (using the Step Transformer) to help improve the long-term sequence modeling in RL (using the Sequence Transformer). Our extensive empirical results show how the learned StAR-representations help our model to outperform the baseline in both Atari and Gym environments, in both offline RL and imitation learning settings. We further demonstrate that our designed architecture and token embeddings are essential to successfully model trajectories, with an emphasis on long-term sequences.

REFERENCES

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pp. 104–114. PMLR, 2020.

Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer, 2021.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding?, 2021.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling, 2021.

Mark Chen, Alec Radford, Rewon Child, Jeff Wu, Heewoo Jun, Prafulla Dhariwal, David Luan, and Ilya Sutskever. Generative pretraining from pixels. 2020.

Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. In *International Conference on Learning Representations*, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Michael Janner, Qiyang Li, and Sergey Levine. Reinforcement learning as one big sequence modeling problem, 2021.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2019.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pp. 1008–1014, 2000.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021a.

Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. Video swin transformer, 2021b.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Daniel Neimark, Omri Bar, Maya Zohar, and Dotan Asselmann. Video transformer network, 2021.

Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*, 2016.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. Citeseer, 1994.

Michael S. Ryoo, AJ Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. Tokenlearner: What can 8 learned tokens do for images and videos?, 2021.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

Gerald Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, volume 139, pp. 10347–10357, July 2021a.

Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. *arXiv preprint arXiv:2103.17239*, 2021b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Jianwei Yang, Chunyuan Li, Pengchuan Zhang, Xiyang Dai, Bin Xiao, Lu Yuan, and Jianfeng Gao. Focal self-attention for local-global interactions in vision transformers. *arXiv preprint arXiv:2107.00641*, 2021.

A Appendix

A.1 Action Prediction Head

For a discrete action space, we use a linear projection $\mathbb{R}^D \to \mathbb{R}^{|\mathcal{A}|}$ to predict $|\mathcal{A}|$ logits corresponding to each action, where $|\mathcal{A}|$ is the size of action space. For a continuous action space, we use a linear layer followed by a $\text{Tanh}(\cdot)$ activation function to map the predictions into a normalized continuous action range of $[-1, 1]$.

A.2 Null Action

We pat the trajectory with null action a_0 when at the initial state when only s_0 is presented in the trajectory. The null action is set as an extra label for discrete action space, like a mask which is an extra vocabulary in language processing. We simply define $a_0 = |\mathcal{A}|$ in Atari, since the original action space is indexed from 0 and this a_0 value is not included in the space, serving as our null action label. For Gym environment, we follow (Chen et al., 2021) using $a_0^i = -10$ for each action dimension. These choices are not specific to our method, just for padding a trajectory, and can be flexibly defined according to each environment.

A.3 Hyper-parameters

We keep most hyper-parameters same as (Chen et al., 2021) for a fair comparison.

Table 6: Our hyper-paremeter settings

Hyper-parameter	Value
Layers	6
MSA heads (long-term)	8
Embedding dimension (long-term)	192
MSA heads (long-term)	4
Embedding dimension (long-term)	64
Nonlinearity	GeLU for self-attention; ReLU for convolution
Dropout	0.1
Learning rate	6×10^{-4}
Adam betas	(0.9, 0.95)
Grad norm clip	1.0
Weight decay	0.1
Learning rate decay	see Chen et al. (2021)