StARformer: Transformer with State-Action-Reward Representations for Robot Learning

Jinghuan Shang, Xiang Li, Kumara Kahatapitiya, Yu-Cheol Lee, *Member, IEEE* and Michael S. Ryoo,

Abstract—Reinforcement Learning (RL) can be considered as a sequence modeling task, where an agent employs a sequence of past state-action-reward experiences to predict a sequence of future actions. In this work, we propose **St**ate-**A**ction-**R**eward Transformer (**StAR**former), a Transformer architecture for robot learning with image inputs, which explicitly models *short-term* state-action-reward representations (StAR-representations), essentially introducing a Markovian-like inductive bias to improve *long-term* modeling. StARformer first extracts StAR-representations using self-attending patches of image states, action, and reward tokens within a short temporal window. These StAR-representations are combined with pure image state representations, extracted as convolutional features, to perform self-attention over the whole sequence. Our experimental results show that StARformer outperforms the state-of-the-art Transformer-based method on image-based Atari and DeepMind Control Suite benchmarks, under both offline-RL and imitation learning settings. We find that models can benefit from our combination of patch-wise and convolutional image embeddings. StARformer is also more compliant with longer sequences of inputs than the baseline method. Finally, we demonstrate how StARformer can be successfully applied to a real-world robot imitation learning setting via a human-following task.

Index Terms—transformer, robot learning, reinforcement learning, imitation learning

1 Introduction

Reinforcement Learning (RL) naturally operates in a sequential manner, wherein an agent observes a state from the environment, performs 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 [1] architectures have become applied to this task as generative trajectory models. 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 [2], [3]. This novel formulation has been shown to be quite useful, especially in terms of its capability to model long-term sequences [3] and sequence distributions [2].

In the state-of-the-art Transformer models for RL such as [2], [3], an input sequence is plainly processed through self-attention – the core modeling component of Transformers [1]. Thus, a given state, action, or reward token may attend to any of the (previous) tokens in the sequence, which allows the model to capture long-term relations. In the case of visual RL, image states are encoded using convolutional networks (CNNs) to create tokens, before processing through self-attention.

- *J. Shang, X. Li, K. Kahatapitiya, and M. S. Ryoo are with the Department of Computer Science, Stony Brook University, Stony Brook, NY, 11794. E-mail: {jishang, xiangli8, kkahatapitiy, mryoo}@cs.stonybrook.edu*
- *Y.-C. Lee is with the Department of Autonomous Vehicle Engineering, Korea Aerospace University, Goyang-si, Gyeonggi-do, 10540, Korea, and also with Artificial Intelligence Laboratory, Electronics and Telecommunications Research Institute, Daejeon, 34129, Korea. E-mail: yclee@kau.ac.kr*

Manuscript received Feb. 16, 2022; revised Aug. 28, 2022; accepted Sep. 4, 2022
(Corresponding authors: Yu-Cheol Lee and Michael S. Ryoo.)

However, tokens within adjacent time steps generally exhibit strong connectivity as a result of potential causal relations. For instance, states in the recent past have a stronger effect on the subsequent action than those in the distant past. Similarly, the immediate-future state and the corresponding reward are the direct results of the current action. In the extreme case of the Markov Decision Process (MDP), which is used to formulate RL problems, the relations are far stronger and more restricted (see Fig. 2). In the aforementioned scenarios, a Transformer naively attending to all tokens naively may suffer from excess information or dilute the truly-essential relation priors, thus making the learning-process harder. This is especially critical when the input sequences are significantly large, either spatially [4] or temporally [3] dimension, and when Transformer models become heavy, i.e., contain a large number of layers [5]. Moreover, the tokenization of overall image states (as a whole) based on CNNs further restricts Transformer models from capturing detailed spatial relations, resulting in the loss of poetentially critical information, particularly in RL tasks with fine-grained regions of interest.

To alleviate these issues, we propose the **St**ate-**A**ction-**R**eward Transformer (**StAR**former), a Transformer architecture that learns **St**ate-**A**ction-**R**eward-representations (i.e., **StAR**-representations) for visual RL. StARformer consists of two basic components: a Step Transformer and a Sequence Transformer. The Step Transformer learns local representations (i.e., StAR-representations) based on self-attention across state-action-reward tokens *within a window of a single time-step*. When learning StAR-representations, we use ViT-like [6] patch-wise embeddings of image states to retain fine-grained spatial information. The Sequence Transformer then combines StAR-representations with pure image state repre-

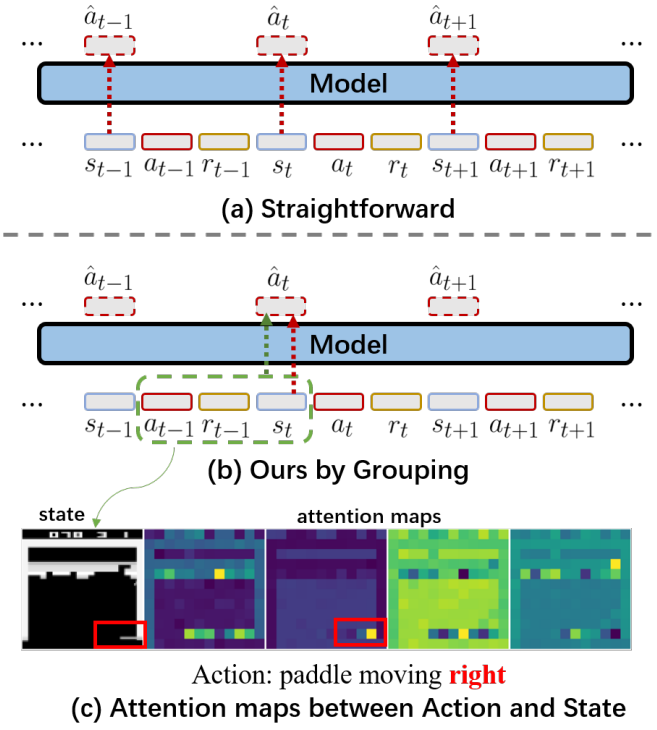


Fig. 1. Illustration of RL as a sequence modeling task using Transformer: (a) A straightforward approach and, (b) Our proposed improvement. Our intuition is to explicitly use local features to facilitate long-term sequence modeling. Red arrows indicate actions being *autoregressively-generated* based on a state, while also considering previous tokens. Green arrows denote our explicit local representation (i.e., grouping), which further improves action generation, combined with the existing state representations. Attention maps between action and pixel state patches produced by our method (in Breakout environment) are visualized in (c). Different regions-of-interest (i.e., ball, paddle, and lower-level bricks) are focused upon each attention head. In the second attention map from the left, weights in the paddle region are directed towards right (circled in red), corresponding to the semantic meaning of the "right" action.

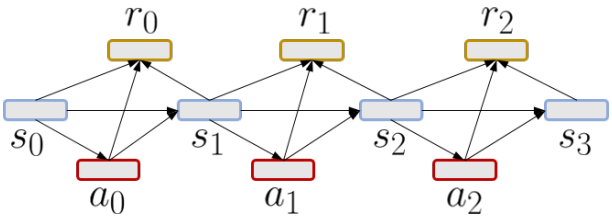


Fig. 2. MDP view of an RL process. Only the connected pairs (denoted by directed arrows) are causally related, whereas all other elements are mutually independent. For instance, state s_1 only depends on previous state s_0 and action a_0, action a_1 is taken based on state s_1, and no direct relations exist between action a_0 and reward r_2.

sentations *from the entire sequence* to generate action predictions. Pure state representations are convolutional features of the image states. This modeling approach can explicitly capture strong short-term relationships while accounting for the overall trajectory. 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. We also find that multiple different state embeddings (such as patch-wise and convolutional) can yield better representations. Finally, we deploy our method on an actual robot which performs a human-following task, to demonstrate its potential in real-world imitation learning settings.

Our contributions are as follows: (1) we model single-step transitions explicitly, relieving model capacity to improve focus on long-term relations; (2) we use both local and global information (separately and combined) to tackle long-term sequence modeling, and (3) we verify its applicability in real-world robot learning.

2 Related Work

2.1 Reinforcement Learning to Sequence Modeling

Reinforcement Learning (RL) is typically modeled as a Markov Decision Process (MDP), where actions are made solely based on the current state according to the Markov property. Accordingly, single-step value-estimation methods have been derived from the Bellman equation, including Q-learning [7] and Temporal Difference (TD) learning, along with many other variants such as SARSA [8], TD(λ) [9], TD-Gammon [10], and Actor-Critic [11]. In recent studies, neural networks have been used to approximate the value function in value-based methods, introducing Deep Q-learning [12].

More recent directions [2], [3] formulate RL as a sequence modeling task, where the model uses a sequence of recent experiences, including state-actions-reward triplets, to predict future actions. This can potentially replace value-estimation methods, and can be trained in a supervised learning manner. However, this approach requires pre-existing trajectories (collected in advance), making it more compliant with offline RL and imitation learning settings. Zheng et al. [13] adapted sequence modeling formulation from offline-settings to online settings. Furuta et al. [14] extended DT [2] to a generalized version to match any given hindsight information.

2.2 Transformers

Transformer architectures [1] were first introduced in language processing tasks [15], [16], [17] to model interactions within a sequence of word embeddings, or, more generally, unit representations or *tokens*. More recently, Transformers have been employed in vision tasks with the key idea of breaking down images/videos into tokens [6], [18], [19], [20], often outperforming convolutional networks (CNNs) in practice. Transformers have also been successfully used to handle sensory information [21] and perform one-shot imitation learning [22]. Chen et al. [2] explored how GPT [17] can be applied to RL in a sequence modeling setting.

Sequence modeling in visual RL is similar to video-based learning in terms of the input data, which consists of observed image (i.e. state) sequences. One challenge in applying Transformers to videos is the large number of input tokens required to be processed. This problem has been investigated in multiple directions, including attention approximation [23], [24], [25], separable attention in different dimensions [18], [26], reducing the number of tokens using local windows [27], [28], and adaptively generating a small number of tokens [29].

StARformer presents a similar concept of using local windows for self-attention in Swin Transformer [27]. Our

approach is also closely related to "divided space-time attention" in [26] and "factorized self-attention" in [18], in terms of handling spatial and temporal attention separately in mentioned literature. Because these methods are designed to reduce the number of tokens, our primary goal is to model short- and long-range contexts separately, thus ensuring higher performance on RL trajectories. Our Step-to-Sequence connection is inspired by [30], which was designed for image domain. However, our model is still unique as (a) "spatial" (using Step Transformer) and "temporal" (using Sequence Transformer) attention are performed in multiple Transformer layers, (b) separate sets of tokens with different origins are used, and (c) a set of locally-attended tokens is fed to perform self-attention in conjunction with a set of global tokens for long-term sequence modeling. Furthermore, we find similar concepts for the use of both attention and convolutional features in [31].

3 PRELIMINARY

3.1 Transformer

Transformer [1] architectures have diverse applications in language [15] and vision tasks [6], [18]. Given a sequence of input tokens $X = \{x_1, x_2, ..., x_n\}$, where $\forall x \in X, x \in \mathbb{R}^d$, a Transformer layer maps the sequence to an output sequence of tokens $Z = \{z_1, z_2, ..., z_n\}$, where $\forall z \in Z, z \in \mathbb{R}^d$ through Multi-head Self-Attention (MSA) [15], followed by Multi-Layer Perceptron (MLP) blocks with a residual connection [32] and Layer Normalization (LN) [33]:

$$\begin{aligned} Z' &= \text{MSA}(\text{LN}(X)) + X \\ Z &= \text{MLP}(\text{LN}(Z')) + Z'. \end{aligned} \tag{1}$$

A Transformer model is obtained by stacking multiple such layers. The mapping for each layer (l) is denoted by $F(\cdot)$: $Z^l = F(Z^{l-1})$. We use the notation $F(\cdot)$ to represent a Transformer layer in the remaining sections.

Self-attention [1], [34], [35], [36] is the core Transformer component that models pairwise relations between tokens. As presented in [1], an input token representation X is linearly mapped into query, key and value representations — that is $\{Q, K, V\} \in \mathbb{R}^{n \times d}$ respectively — to compute self-attention as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d}})V. \tag{2}$$

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

Vision Transformer (ViT) [6] extends the same concept of self-attention to the image domain by mapping a set of non-overlapping image patches to a sequence of tokens using a fully-connected (FC) layer. 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 tokenization approach as a part of our state token embeddings.

3.2 RL as Sequence Modeling

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:

$$\tau = \{s_1,\ a_1,\ r_1,\ s_2,\ a_2,\ r_2,\ \ldots,\ s_t,\ a_t,\ r_t\}. \tag{3}$$

RL is formulated as a sequence modeling task by considering action predictions from past experience [2], [3] according to the following equation:

$$Pr(\hat{a}_t) = p(a_t|\ s_{1:t},\ a_{1:t-1},\ r_{1:t-1}). \tag{4}$$

This novel scheme does not employ conventional value estimation or policy gradient methods.

A recent study [2], [3] attempted to use an existing Transformer architecture [17] for RL with the aforementioned formulation. In [2], [3], the states (s), actions (a), and rewards (r) are considered as input tokens (see Fig. 1a), and a causal mask is used to ensure an autoregressive output sequence generation (that is, Eq. 4). Here, a token could access any of its corresponding prior tokens—in time— through self-attention.

In contrast, our formulation attends tokens with (potentially) strong causal relations *explicitly*, while also attending to long-term relations as well. To achieve this, we split each trajectory into small groups of state-action-reward tuples (i.e., s, a, r) to learn local relations within the tokens of each group through self-attention (see Fig. 1b). We then model long-term relations in conjunction with the learned local relations. Our grouping is based on the intuition that the local causal relations between s, a and r are strong; that is, the reward r_{t-1} and the state s_t are direct results of the action a_{t-1}. Consequently, StARformer learns strong intermediate representations from local groups of $(a_{t-1},\ r_{t-1},\ s_t)$ explicitly, to facilitate long-term sequence modeling. During the learning of local representations, we further split each image state into patches when learning local representations to more effectively capture fine-grained relations between the action and local state regions. We find that our scheme is similar to the divided spatial-temporal attention in [26], which is applied in the video domain with the intuition of reducing computation requirements.

4 STARFORMER

4.1 Overview

StARformer consists of two basic components: Step Transformer and Sequence Transformer, together with interleaving connections (see Fig. 3). Step Transformer learns StAR-representations from strongly-connected local tokens *explicitly*, which are then fed into the Sequence Transformer along with pure state representations to model the whole input trajectory. At the output of the final Sequence Transformer layer, we make action predictions via a prediction head. In the following subsections, we will introduce the two Transformer components, and their corresponding token embeddings in detail.

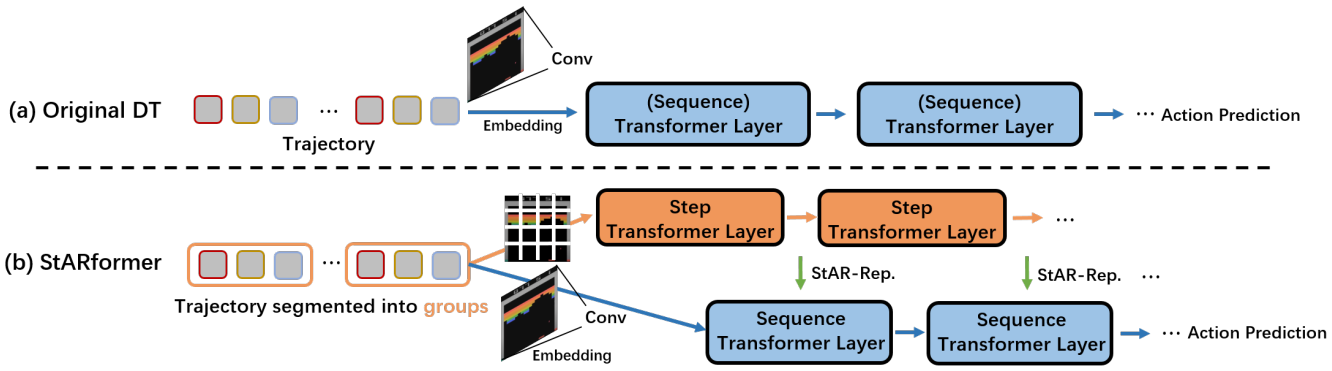


Fig. 3. (a) Structure summary of original DT [2], where its Transformer layers act similar as our Sequence Transformer. (b) StARformer consists of Step Transformer and Sequence Transformer, to separately model a single-step and the sequence as-a-whole, respectively. Two types of layers are connected at each level via learned StAR-representations. In terms of state embedding methods, DT uses only convolution, while StARformer uses ViT-like [6] embeddings (patches) in Step Transformer and convolution in Sequence Transformer separately.

4.2 Step Transformer

4.2.1 State-Action-Reward Embeddings

Grouping state-action-reward: Our intuition of grouping is to model strong local relations explicitly. To do so, we first segment a trajectory (τ) into a set of groups, each of which consists of a previous action (a_{t-1}), corresponding reward (r_{t-1}), and current state (s_t)[1] (see Fig. 4). Each element within a group has a strong causal relationships with the other elements.

Patch-wise state token embeddings: In Step Transformer, each input image state is tokenized by dividing it into a set of non-overlapping spatial patches along its spatial dimensions, in accordance with ViT [6]. Consider an image state $s_t \in \mathbb{R}^{H\times W\times C}$, we divide it into patches $\{s_t^i\}, \in \mathbb{R}^{h\times w\times C}$, which generates a total of HW/hw state tokens per image. We use a linear projection, where the weights of which are shared by each state patch across all groups, to create token embeddings ($\mathbb{R}^{hwC} \to \mathbb{R}^d$ for image states or $\mathbb{R}^1 \to \mathbb{R}^d$ for vector states) according to the following equation:

$$z_{s_t^i} = \text{FC}(\text{Flatten}(s_t^i)) + e_i^{\text{spatial}} \tag{5}$$

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

Our motivation for using patch embeddings is to create fine-grained state embeddings that allows the Step Transformer to model the relations between actions and rewards within local state regions. This is especially useful in RL tasks, where local regions of interest can be critical. We empirically validated this in our experiments (Sec 6.5.1).

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, as in [2]. Thus, each token is mapped to an appropriate value range for robot control.

$$z_{a_{t-1}} = \text{FC}(a_{t-1}), z_{r_t} = \text{Tanh}(\text{FC}(r_t)) \tag{6}$$

1. We pad the trajectory with a null action and zero reward at the initial state s_1 of the trajectory (see Section 4.4)

Altogether, we obtain a state-action-reward representation as the input to the initial Step Transformer layer, expressed as:

$$Z_t^0 = \{z_{a_{t-1}}, z_{r_t}, z_{s_t^1}, z_{s_t^2}, \ldots, z_{s_t^n}\}. \tag{7}$$

We obtain T groups of such token representations per trajectory that are simultaneously processed by the Step Transformer with shared parameters.

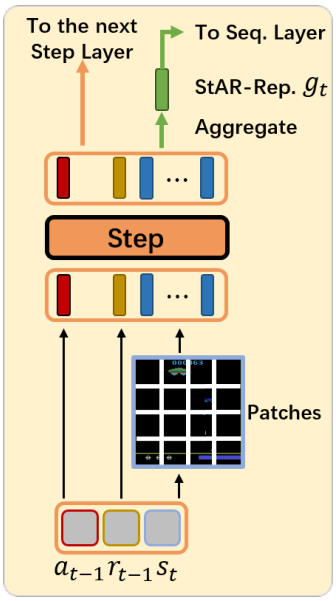


Fig. 4. Overview of Step Transformer. Output tokens are (1) sent to the next Step Transformer layer and (2) aggregated to produce StAR-representation.

4.2.2 Step Transformer Layer

We adopt the conventional Transformer design from [1] (see Section 3.1) 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:

$$Z_t^l = F_{\text{step}}^l(Z_t^{l-1}). \tag{8}$$

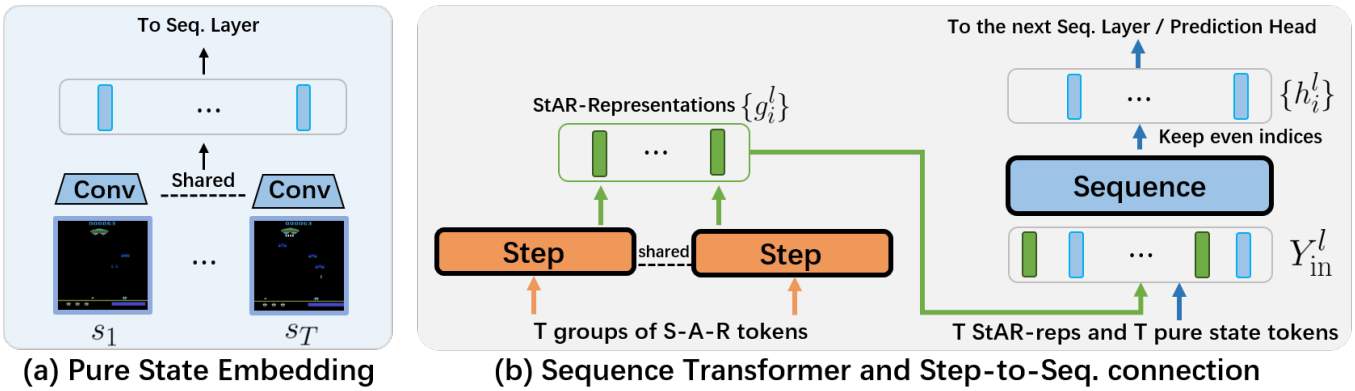


Fig. 5. (a) Pure state embeddings are learned from shared convolutional layers. (b) Sequence Transformer takes StAR-representation and the pure state tokens and generate output tokens.

Each Step Transformer layer l outputs a State-Action-Reward-representation (StAR-representation) $g_t^l \in \mathbb{R}^D$ by aggregating the output tokens $Z_t^l \in \mathbb{R}^{n\times d}$ (see the green flows in Fig. 4). We concatenate the 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.

$$g_t^l = \text{FC}([Z_t^l]) + e_t^{\text{temporal}} \tag{9}$$

Here $[\cdot]$ represents the token concatenation within each group, and $e_t^{\text{temporal}} \in \mathbb{R}^D$ represents the temporal positional embeddings for each timestep. Note that we add these temporal positional embeddings to g_t^l at each layer because $[Z_t^l]$ is learned agnostically with respect to time. Finally, the output StAR-representation g_t^l is fed into the corresponding Sequence Transformer layer for long-term sequence modeling.

4.3 Sequence Transformer

Our Sequence Transformer (illustrated in Fig. 5) models long-term sequences by looking at the learned StAR-representations and the *pure state tokens* (introduced below) over the entire trajectory.

4.3.1 *Pure State Token Embeddings*

In contrast to the patch-wise token embeddings in Step Transformer, we embed the overall input image state s_t to generate pure state tokens h_t^0. Each such token represents a single-state representation that describes the state globally in space. We achieve this by processing each state through a CNN encoder, whose convolutional layers spatially combine the features. The convolutional encoder is obtained from [37].

$$h_t^0 = \text{Conv}(s_t) + e_t^{\text{temporal}}, \tag{10}$$

where $e_t^{\text{temporal}} \in \mathbb{R}^D$ represents the temporal positional embeddings exactly the same as what we add to g_t.

4.3.2 *Sequence Transformer Layer*

As in Step Transformer, we use the conventional Transformer layer design from [1] 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 pure 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 follows:

$$Y_{\text{in}}^l = \{g_1^l,\ h_1^{l-1},\ g_2^l,\ h_2^{l-1},\ \ldots,\ g_T^l,\ h_T^{l-1}\}. \tag{11}$$

We place g_t^l before h_t^{l-1}, which originates from s_t, because g_t^l contains information pertaining to 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 to any future tokens (i.e., $> t$).

Sequence Transformer computes an intermediate set of output tokens as follows:

$$Y_{\text{out}}^l = F_{\text{sequence}}^l(Y_{\text{in}}^l). \tag{12}$$

We then select the tokens at even indices of Y_{out}^l (where indexing starts from 1) as the pure state tokens h_t^l, which are fed into the next Sequence Transformer layer. Because the even indices correspond to the tokens originating from the pure state representations s_t (Fig. 5(b)), they should be used to *predict* actions from an autoregressive perspective (see Fig. 1). In contrast, any tokens in Y_{out}^l with odd indices do not propagate to the next layer.

$$\begin{aligned} Y_{\text{out}}^l &= \{y_{\text{out};1}^l,\ y_{\text{out};2}^l,\ \ldots,\ y_{\text{out};2T}^l\} \\ h_i^l &:= y_{\text{out};2i}^l \end{aligned} \tag{13}$$

Therefore, the overall input (StAR-representation and pure state representation) to the subsequent layer can be expressed as (by rewriting Eq. 11):

$$\begin{aligned} Y_{\text{in}}^{l+1} &= \{g_1^{l+1},\ h_1^l,\ g_2^{l+1},\ h_2^l,\ \ldots,\ g_T^{l+1},\ h_T^l\} \\ &= \{g_i^{l+1},\ y_{\text{out};2i}^l\}_{i=1}^T. \end{aligned} \tag{14}$$

4.3.3 *Action Prediction*

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

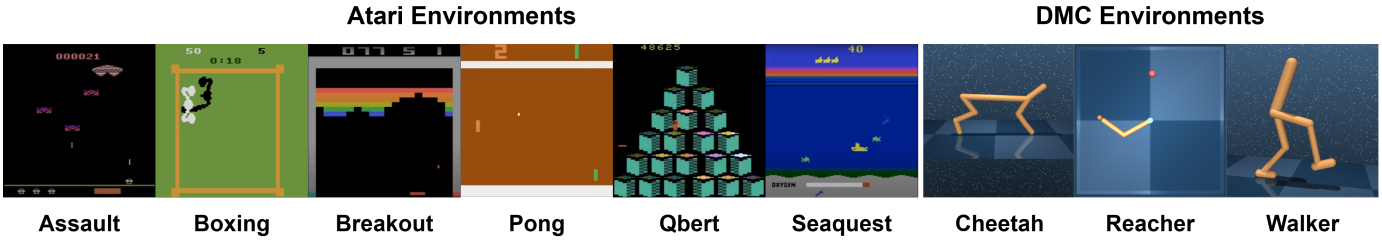


Fig. 6. Environments used in our experiments: Atari with a discrete action space, and DMC with a continuous action space. As in prior studies, we convert RGB images to grayscale images as the model input.

4.4 Training and Inference

During training, we use trajectories τ with a length T, randomly sampled (and sliced) from a memory buffer. Ground-truth actions are used as labels. Cross-entropy loss is used for a discrete action space, whereas the mean-squared error (MSE) is used 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 the zero reward which padded at the start of each trajectory. StARformer makes an initial prediction $\hat{a}_1$ based on τ, and receives the next state s_2 and reward r_2. We append these $\hat{a}_1$, s_2 and r_2 to the trajectory τ and repeat this procedure until the end of one RL episode.

5 Simulation Experiment Settings

5.1 Settings

We use offline RL [38] and imitation learning as our experiment settings. These settings are commonly used in related studies that formulate RL as a sequence modeling task [2], [3], 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 than conventional RL [38] due to the shifted distribution.

Imitation learning, however, is the setting that the agent is not exposed to reward signals. Therefore, we simply remove the rewards from the dataset used in offline RL. This presents a significant challenge compared to traditional imitation learning because the provided trajectories are sub-optimal (i.e., with lower rewards than true experts). The only difference in terms of the model structure is that there is now one less reward token in the input of Step Transformer in our model, and T less reward tokens in the baseline model (T is the number of time-steps in the input trajectory).

5.2 Environments

We use image-based Atari [39] (discrete action space) and DeepMind Control Suite (DMC) [40] (continuous action space) to evaluate our model's performance in different types of tasks, as shown in Fig. 6 with image examples. We select six games from Atari: Assault, Boxing, Breakout, Pong, Qbert, and Seaquest. As in [2] we use 1% (500k steps) of the DQN replay buffer dataset [41] to perform a thorough and fair comparison. We select five continuous control tasks in DMC [40]: Ball-in-cup-catch, Cheetah-run, Finger-spin, Reacher-easy, and Walker-walk. In DMC, we collect a replay buffer (i.e. sub-optimal trajectories) generated by training an SAC [42] agent from scratch for 500k steps in each task. This is a similar setting to "Medium-Replay" in the D4RL [43] dataset used by DT [2]. Note that these continuous control tasks involve image inputs, which the previous study [2] did not cover (originally using Gym [44]). Furthermore, learning continuous control from pixels is more challenging than directly learning from actual joint states [45], and is generally harder than Atari games due to the larger continuous action space (eg. six dimensions in Cheetah and Walker). We report the absolute value of episodic returns (i.e., cumulative rewards). The results are averaged across multiple randomly initialized runs (7 seeds in Atari and 10 in DMC), each run is evaluated by 10 randomly initialized episodes.

2. We use an additional action to represent null action in the discrete action space, and a zero vector in the continuous action space.

5.3 Baselines

We select Decision-Transformer (DT) [2], a SOTA Transformer-based sequence modeling method for RL. Although Trajectory-Transformer [3] exsists, it is not designed for image inputs. We use most of the same hyper-parameters used in original DT [2] for the Atari environments, without extra tuning (details in Appendix Tables 4 and 5). Because DMC environments are not covered by DT [2], we carefully tune the baseline first and then use the same set of hyper-parameters in our method. We also compare with the SOTA non-Transformer offline-RL methods, including CQL [46], QR-DQN [47], REM [48] in Atari, and BEAR [49], IQL [50], and TD3+BC [51] in DMC. For imitation (behavior cloning), we only compare with DT [2] and straightforward behaviour cloning with ViT (denoted as BC-ViT).

6 Results on Simulation Environments

6.1 Improving Sequence Modeling for RL

We first compare our StARformer (StAR) with the state-of-the-art Transformer-based RL method in Atari and image-based DMC environments, under both offline RL and imitation learning settings. We select the Decision-Transformer proposed in [2], (referred to as DT) as our baseline. Here, we maintain $T = 30$ for all environments, which is the number of time-steps (length) of each input trajectory (τ). We also

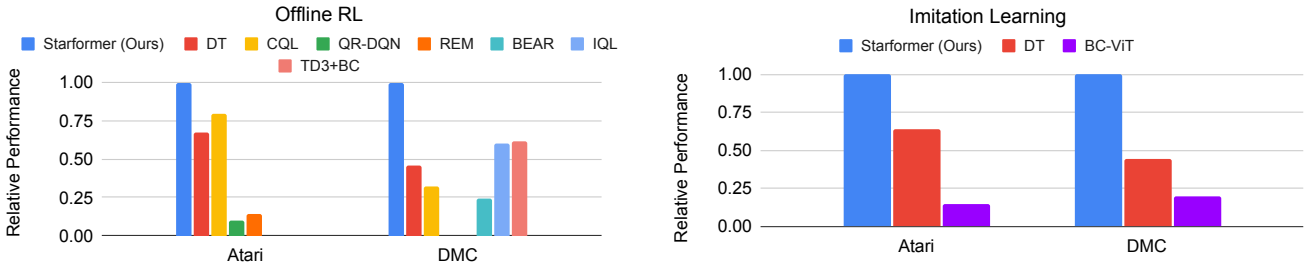


Fig. 7. Relative performance of episodic returns. We compare StAR with DT [2] in both offline-RL and imitation settings. We also compare with previous offline-RL methods in their capable environments, that is, CQL [46], QR-DQN [47] and REM [48] in Atari (discrete action space), and CQL [46], BEAR [49], IQL [50], TD3+BC [51] in DMC (continuous action space). We introduce a baseline method BC-ViT for imitation learning, which performs behavior cloning using a ViT encoder naively. StAR consistently outperforms others in both settings. Here, the results are averaged across all environments and random seeds (7 in Atari and 10 in DMC), and normalized w.r.t. the performance of StAR. Please refer to Table 1 in the Appendix for absolute values corresponding to the above results, with more comparisons of offline-RL methods.

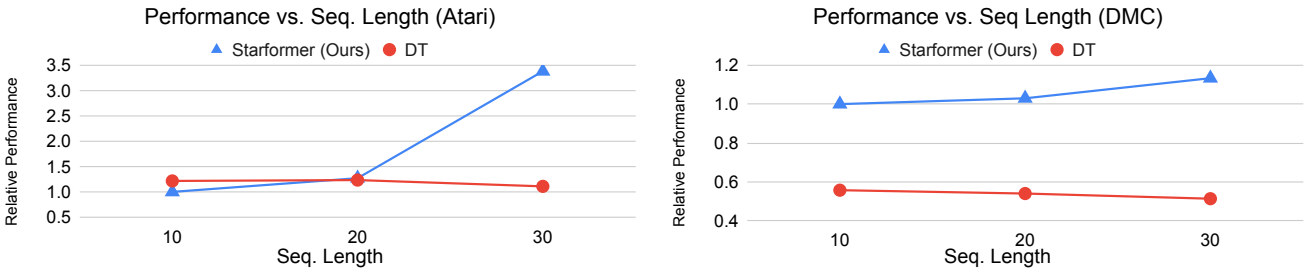


Fig. 8. Change in performance with the length of input sequence, $T \in \{10, 20, 30\}$, in Atari and DMC (averaged across tasks). Here, the results are normalized w.r.t. StARformer performance at $T = 10$. We evaluate with offline-RL, as DT is more competitive in this setting. The results show a performance gain in StARformer with longer input sequences, whereas DT [2] exhibits a drop in performance. This validates the superior performance of our method in long-term sequence modeling. Please refer to Appendix Fig.1 for a more detailed comparison.

compare our method to CQL [46], a SOTA non-Transformer offline-RL method. Fig. 7 shows that our method outperforms the baselines in both offline RL and imitation learning settings, which suggests that our method models image sequences more effectively.

6.2 Scaling-up to Longer Sequences

In this experiment, we evaluate how StARformer and DT perform with different input sequence lengths, specifically $T = \{10, 20, 30\}$. In Fig. 8, we see that StARformer achieves satisfactory performance with longer trajectories, whereas DT exhibits saturation as early as $T = 10$. This validates our claim that considering short-term and long-term relations separately (and then fusing) helps the model scale-up to longer sequences. Instead of learning Markovian pattern attentions [3] implicitly, we model it explicitly in our Step Transformer. This acts as an inductive bias, relieving the capacity of Sequence Transformerto place more focus on long-term relations. In contrast, DT uses the off-the-shelf language model GPT [17], that does not consider the Markov property. Although GPT [17] structure performs well for long sentences, we show here it does not adequately handle long RL sequences with image states.

6.3 Reward setting: Return-to-go, stepwise reward, or no reward at-all?

We also intend to determine how different reward settings affect sequence modeling, specifically, return-to-go

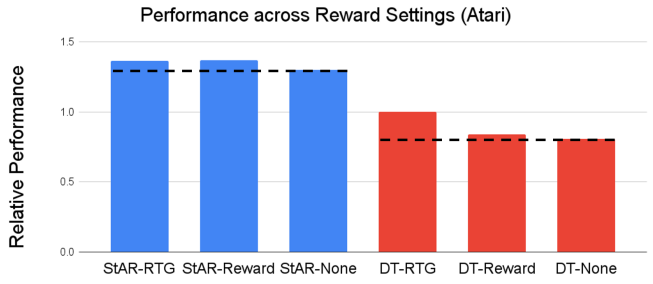


Fig. 9. Performance in different reward settings: return-to-go (RTG), stepwise reward, and no reward at-all (labeled as 'None') settings. StARformer performs similarly in the two settings with reward, and demonstrates slight degradation when rewards are not provided. In contrast, DT shows a higher performance when used with RTG, compared to DT under either stepwise or no reward settings. Results are averaged across six Atari environments and normalized w.r.t DT-RTG.

(RTG) [2], stepwise reward, and no reward at-all. Decision-Transformer [2] originally uses RTG $\hat{R}_t$, which is defined as the sum of future step-wise rewards: $\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$. Similar concepts have been studied in hindsight-related methods to facilitate RL tasks [52], [53], [54], [55], [56], [57], [58]. Stepwise reward r_t is the immediate reward generated by the environment in each step, which is generally used in most RL algorithms. Our StARformer uses step-wise reward as the default, guided by the motivation of modeling single-step transitions. The no reward setting corresponds to an imitation formulation.

The results for each reward setting are presented in Fig. 9. StARformer and DT behave differently under various reward seetings. Although both methods show an increase in performance when a reward is provided, StARformer performs similarly regardless of RTG or stepwise reward, whereas DT relies more on RTG. Note that in the setting with no reward at-all, the rest of the dataset (i.e. states and actions) is still the same. Sequence modeling can still work on state-action trajectories without a reward when the model has sufficient capacity to mine relevant patterns and generate better representations. This suggests that StARformer without a reward can outperform DT with RTG.

6.4 Visualization

We present the attention maps between action and state patches in Step Transformer(see Fig. 10) at several timesteps extracted from a trajectory in Breakout. In this game, the agent should move the paddle to bounce the ball back from the bottom, after the ball falls down while breaking the bricks on the top. In the presented attention maps, the regions with a high attention score (highlighted) mainly overlap with the locations of the ball, paddle, and potential target bricks. We find the attention maps corresponding to Head #1 are particularly interesting. Here, the focused regions corresponding to the paddle show a directional pattern, associated with the semantic meaning of the actions "moving the paddle right", "left", or "stay". This affirms that Step Transformer captures the essential spatial relations between actions and state patches, which is important for decision making. Moreover, in Head #2, we observe that the focused regions correspond to the locations of the ball, except when the ball is outside of the boundary, too-close to the paddle or indistinguishable within the bricks. Overall, these attention maps suggest that how our model obtains a basic understanding of the Breakout game.

6.5 Ablations

StARformer has two major differences compared to the baseline method DT [2]: (1) it learns two types of representations, namely, StAR-representations and pure state embeddings; and (2) it employs a separate Step Transformer to merge these two types of embeddings. The following sections present ablation studies on each of these distinctions.

6.5.1 StAR-representations and pure state embeddings

In terms of the learned representations, our approach deviates from the baseline method, which models all the tokens from the trajectory (CNN-extracted states, actions, and rewards) directly, whereas we:

- use ViT-like [6] patch-wise image embeddings when learning StAR-representations in Step Transformer, and,
- consider pure state representations h_t (from convolutional layers) together with StAR-representations in Step Transformer.

To investigate the effect of the above changes, we vary the state embedding methods used to learn s_t in Step Transformer and h_t in Sequence Transformer. Specifically, we

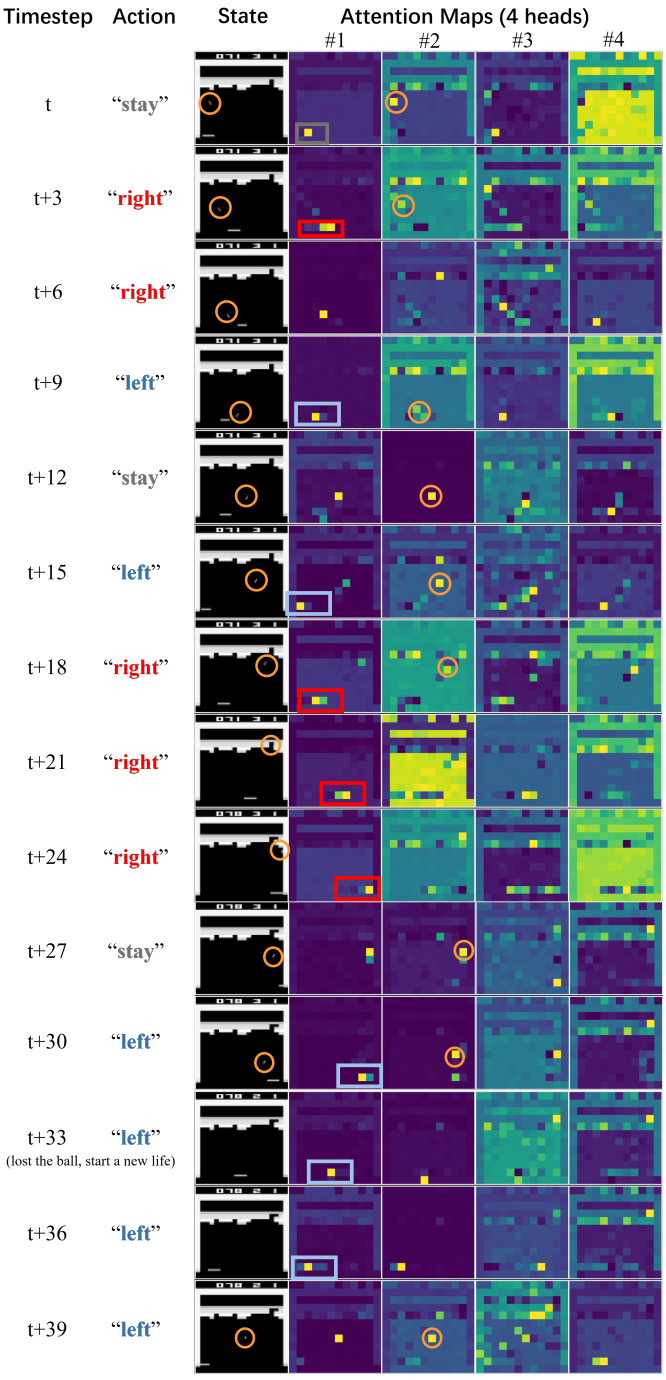


Fig. 10. Visualization of attention maps in our Step Transformer , extracted from a clip of a Breakout trajectory, with all corresponding timesteps and actions annotated. Attention weights are computed between the action token and state patch tokens. We highlight the ball in orange for convenience. Actions and corresponding regions in attention maps were labeled in different colors ("left" (blue), "right" (red) and "stay" (gray)). In general, we find high attention scores in the actual locaitons of the paddle and the ball. Each attention head focuses on different input elements: bricks, the ball or the paddle. Specifically, we find a clear and consistent semantic meaning in Head #1, which gives higher attention scores with a directional pattern to the right when the action is "right". When the action is "stay", it focuses on the paddle or the ball itself. Head #2 mostly focuses on the ball, unless the ball is lost (out of boundary) or indistinguishable within the bricks.

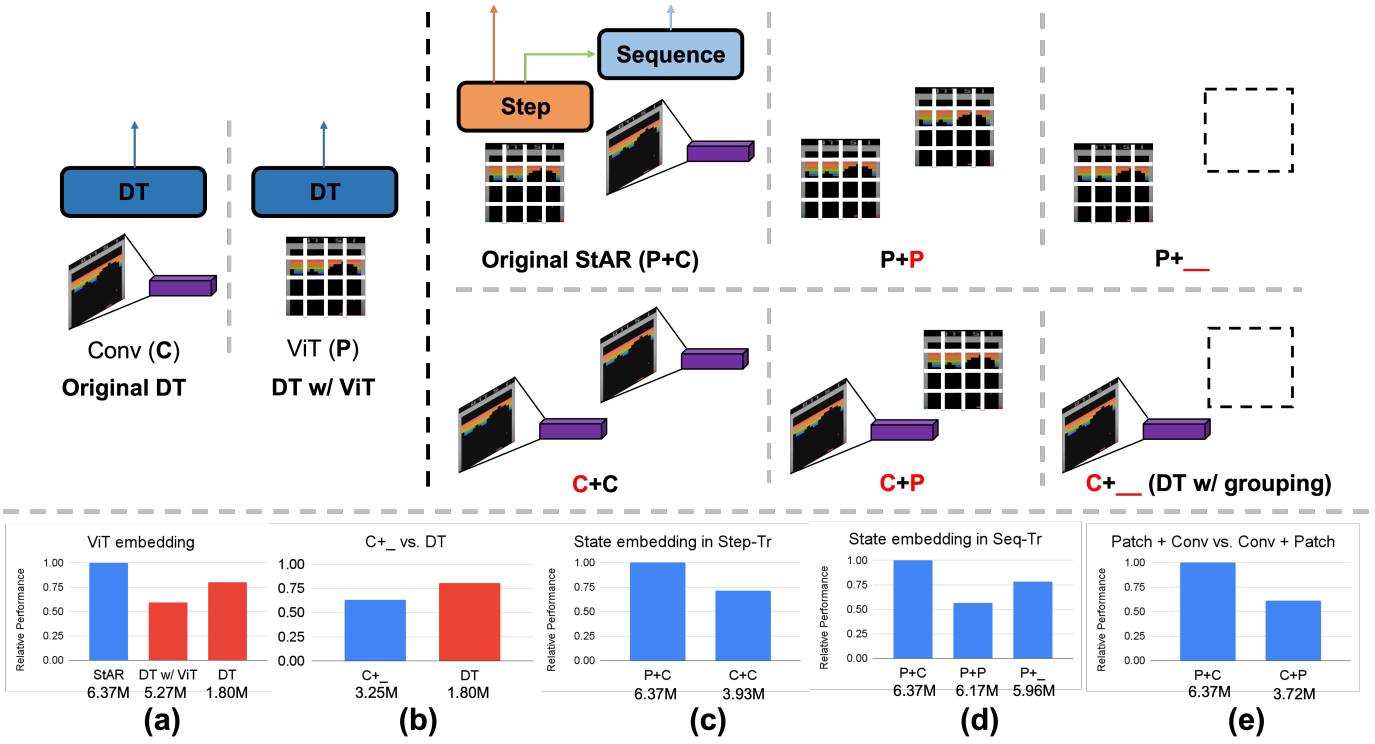


Fig. 11. (Top): Embedding methods used in the original DT, StARformer (StAR), and their variants. We label ViT (patch embeddings) as **P**, convolution as **C**, and None (not using the corresponding embedding) as "__". (Bottom): (a-e) performance comparisons between variants. Relative performance (in offline-RL, averaged across Atari tasks) in all comparisons is normalized w.r.t. StAR. We attach the number of parameters (in millions (M)) of each method below each bar chart. These observations validate that (1) StAR benefits from the fusion of ViT and convolutional features; (2) ViT performs better in Step Transformer and convolution works better in Sequence Transformer due to the specific context (short-term and long-term, respectively). Specific results for each task are listed in Appendix Table 2.

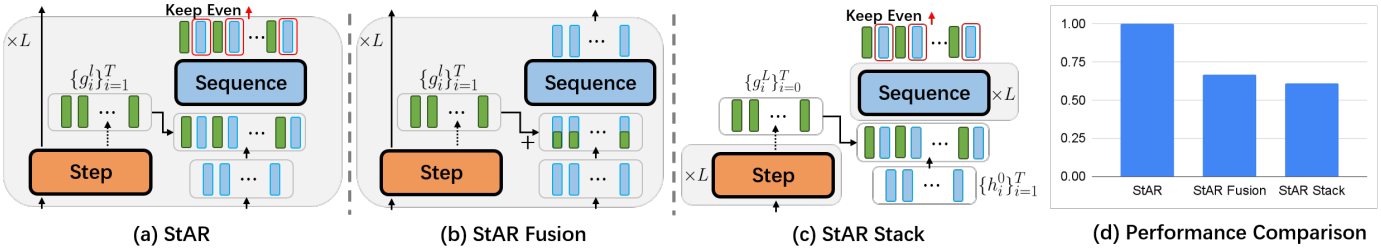


Fig. 12. Illustration of our ablation study on Transformer connection of our model. Two variants — (b) StAR Fusion and (c) StAR Stack — are shown in comparison to our original (a) StAR model. g^l (green) is the StAR-representation from l-th layer of the Step Transformer and h^0 (blue) denotes the initial pure state embeddings. L is the number of layers. Positional embeddings are omitted for simplicity. (d) Experiments (offline-RL) find that the original structure (StAR), which is a layer-wise fusion, yields higher performance than StAR Fusion and StAR Stack connections, as seen with higher rewards. Please refer Table 3 in the Appendix for per-task results.

consider the following: (1) ViT features (patch embeddings, labeled as **P**), (2) convolutional features (labeled as **C**), and (3) none (not having the corresponding embedding, labeled as "__"). The original StARformer can be represented as **P**+**C** (patch embeddings for s_t and convolutional embeddings for h_t). Other variants include: **P**+**P**, **P**+__, **C**+**P**, **C**+**C**, and **C**+__. We also implement a variant of DT using ViT for state embedding (denoted as DT w/ ViT), to match our method in terms of having a similar embedding method and capacity (6.4M parameters vs. 5.3M parameters in our method).

When comparing StARformer with the original DT and DT w/ ViT (see Fig. 11(a)), we observe a performance drop in DT when combined with ViT, which suggests that replacing convolutional features with ViT-like features naively does not benefit the model despite the increased capacity (similar to ours). StARformer, however, does not benefit only from the larger capacity, but also from its better structural design, as verified in following experiments (see Fig. 11(c)(d)(e)). From Fig. 11(b), we find that **C**+__ which only uses convolutional features at Step Transformer, performs worse than DT. This is because convolutional features are highly abstracted, making them unsuitable for single-step transition (i.e., fine-grained) modeling.

Comparing **P**+**C** with **C**+**C** (Fig. 11(c)), the poorer performance of **C**+**C** suggests that patches embeddings are better suited for modeling single-transitions in Step Transformer. In Fig. 11(d), we compare **P**+**C** with **P**+**P** and **P**+__. We find

that convolutional features work best in Sequence Transformer, which indicates that they provide abstract global information that is useful for long-range modeling (coarse), in contrast to patched embeddings. The observations from these comparisons of StARformer variants suggest that our method benefits from the fusion of patches and convolutional features. This is further evaluated this by comparing **P**+**C** with **C**+**P** (Fig. 11(e)), where **P**+**C** performs better, confirming the "fine-grained (patches) to high-level (conv)" fusion method as a best match with our sequence modeling scheme of "single-transition followed-by long-range-context".

In summary, these observations supports two motivations on our model design (using **P**+**C**). (1) StARformer benefits from using **P** and **C** to encode a visual state from *different aspects* — **P** captures local, spatial details and **C** captures global state information. Other variants using only one kind of embeddings are relatively limited to represent a state. (2) Furthermore, the two kinds of embeddings are used *in appropriate modules*. **P** is used in Step Transformer to better fuse with action and reward tokens, and **C** is used in Sequence Transformer in order to summarize states at high level for easier temporal modeling. The variant that exchanges the usage of embeddings (**C**+**P**) has been shown worse than the desired usage.

6.5.2 Step-to-Sequence Layer-wise Connections

In our model, we use learned StAR-representations g and pure state representation h together in the Sequence Transformer, implemented as layer-wise connections. In fact, 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 investigate why such layer-wise connections are important by comparing with two other variants: (1) g_t^l is fused with h_t^l by summation (referred as StAR Fusion, see Fig. 12(b)), and (2) 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 StAR Stack, see Fig. 12(c)). Results of these configurations are shown in Fig. 12(d). It shows that our layer-wise connection method is important for long-term prediction, which fuses information from StAR-representation in each layer, and is done through attention rather than direct fusion.

7 Real-world Robot Imitation Learning Experiment: Human-following

This section demonstrates how our method can be applied to a ground mobility robot via a human-following task, in the formulation of imitation learning. Our robot system and human-following task are illustrated in Fig. 13. To the best of our knowledge, this is first study to apply sequential modeling methods to real-robot imitation learning.

The reason for this is Sequential modeling (Transformer) can be a cheaper robot imitation learning method that does not requrie online (i.e., real-robot) policy learning, thus conserving time and human efforts. In such a formulation, learning could be done in only two steps, as in behavior cloning: (1) collecting the demonstration dataset, and (2) training the sequential model (policy). This end-to-end training scheme is also promising in the context of robotic tasks, like navigation.

The following subsections introduce our human-following task (in Sec. 7.1) and system settings (in Sec. 7.2), followed by the release of a new real-world human-following dataset (in Sec. 7.3), and our offline and online evaluation results (in Sec. 7.4 and Sec. 7.5, respectively).

7.1 Human-following Task

We set up a human-following task to collect data in the context of robot learning. In this task, the robot agent should follow a moving human target (as long as possible) without any disruptions, such as losing or changing targets, or interfering with other moving people. A similar navigation in crowded environment has been explored by Monaci et al. [59] and is shown to be very challenging. The experimental site is set up at the lobby of a university department building with up to 12 moving or standing people. First, we collect expert demonstrations (the dataset), described in Sec. 7.3. Subsequently, we trained a policy based on the collected dataset. After the convergence of training, the model was ready to be evaluated/used offline or online.

7.2 Robot System Settings

To implement the robot system for the human-following task, we construct a mobile robot platform equipped with a computer, a router, a spherical camera, a LiDAR and a projected texture stereo (PTS) camera, as shown in Fig. 14(a). The spherical camera captures a panoramic image with 3840×1920 resolution at a frequency of 29.97Hz. The PTS camera acquires 3-channel color and depth images with 1280×720 resolution at a frequency of 30Hz. The LiDAR sensor scans 16-channels of distance ranges in the vertical direction with a 2° interval angle for objects within the $100m$ distance.

In terms of software, the sensors, robot platform, and the trained model are integrated using Robot Operation System (ROS) as middleware. Shown by dashed lines and boxes in Fig. 14(b), the panoramic image, RGB-D image, linear and angular velocities, and point cloud data are synchronized to be saved as the dataset. The collected data are used to train the human-following model and evaluate its offline performance. For this, an expert manually controls the linear and the angular velocities of the robot by a wireless game controller, adapting it to the human-following task. Simultaneous localization and mapping (SLAM) algorithm is used to evaluate the model's performance by estimating the accurate positions and velocities of the robot using the point cloud data from LiDAR. An online evaluation of the trained model is subsequently performed, as shown by the solid lines and boxes in Fig. 14(b). We use FOV images as the input. The trained model generates the linear and angular velocities as actions for controlling the robot. The entire system runs at $\sim$ 10Hz during data collection and online evaluation.

7.3 Human-following Dataset

We collected a human-following expert dataset by manually controlling the robot to follow moving human targets.

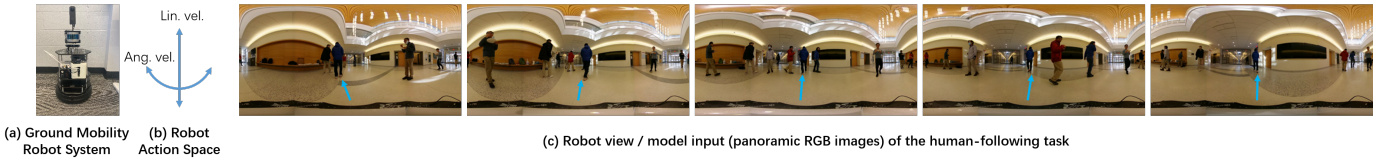


Fig. 13. (a) The ground mobility robot system we use. (b) The action space (control command) of the robot includes linear velocity and angular velocity. (c) A trajectory clip from the robot view (model input) of our human-following task. We highlight the locations of target person by blue arrows. In our human-following task, the robot should keep following the target person when other non-target people are actively walking, standing, and crossing.

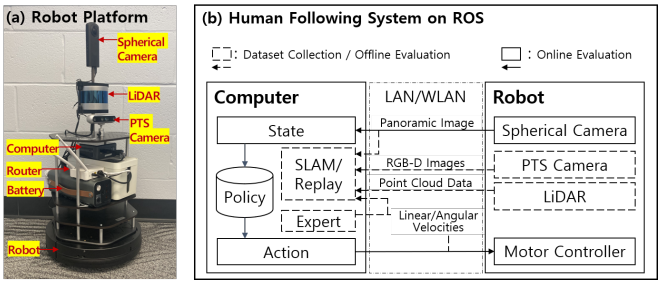


Fig. 14. Overview of our robot system: (a) robot platform configuration and, (b) software schematics and data flow. Here, the dotted lines and boxes represent the dataset collection and offline evaluation, whereas the solid lines and boxes represent the online evaluation. Note the SLAM is only used to annotate robot global positions, which are then used as labels in the position prediction task and trajectory visualization.

The experiment site was set in a department lobby with a group of volunteers[3]. During data collection, the robot was continuously controlled to follow a specific person (target) continuously along a trajectory. Other volunteers were required to walk, stand, and sit randomly at the experimental site. They were also encouraged to cross the robot's path to confuse it or occlude the target.

In terms of brief statistics of the dataset, there were 2-12 volunteers simultaneously in the scene, and we collected nine valid trajectories with the robot following different targets, reaching 51,817 frames or ∼86 minutes in total. We recorded stereo images, RGB-D images, LiDAR data (point clouds in local coordinates), and robot actions (linear and angular velocities). A sample frame (images and point clouds) of the collected data is shown in Fig. 15. Note that our study only considers image modalities as inputs. Point clouds are used to annotate the ground truth of the position prediction task and visualization via a conventional SLAM algorithm.

7.4 Offline Evaluation

7.4.1 Evaluation Task Setup

We perform offline evaluation of multiple visual input modalities and on multiple tasks. As input modalities, we consider (a) SRGB: panoramic images from the spherical camera, (b) FRGB: field-of-view (FOV) RGB images from the PTS camera, and (c) FRGB-D: FRGB input plus depth channel images from the PTS camera. We set up two evaluation tasks: policy learning and 2D position prediction.

3. We provided the volunteers with free food for their contribution. All volunteers were required to wear masks whenever they were not actively eating or drinking.

Policy learning (action prediction): The policy learning task focuses on predicting robot control commands, which include linear and angular velocities. Here, we consider continuous and discrete action space settings. In the continuous action space, evaluations are performed using the mean squared error (MSE) between the predicted $\hat{a}_t$ and ground truth a_t robot controls (normalized to $[-1, 1]$ range) as given by:

$$\text{error} = \frac{1}{N}\sum_{t} ||\hat{a}_t - a_t||_2, \tag{15}$$

where N denotes the trajectory length. In the discrete action space, the original 2D linear-angular space is divided into finite categories. We create two discretized action spaces — 6-action and 10-action spaces — as shown in Table 1. Here, actions consist of combinations of (normalized) linear and angular velocities, and we re-label the original real control commands with the discrete action categories as above. Finally, we use classification accuracy as a metric to evaluate performance.

2D position prediction: The other evaluation task focuses on predicting the 2D position of the robot. We perform this task by predicting displacement $\Delta\hat{p}_i$ (in Cartesian coordinates) at each timestep, and accumulating all predicted displacements until timestep t to obtain the global position $\hat{p}_t$ given by:

$$\hat{p}_t = \sum_{0}^{t} \Delta\hat{p}_t. \tag{16}$$

We measure the Euclidean distance between the predicted position and the ground-truth position at the end of the trajectory as follows:

$$\text{error} = ||\hat{p}_N - p_N||_2. \tag{17}$$

Note that the ground truth 2D positions are obtained from SLAM results. We ignore the z-axis (height) because the robot does not move along that axis.

Cross-validation was performed for all tasks. In each cross-validation run, we use 8 of 9 total trajectories to train and use the remaining trajectory for evaluation. The evaluation results are averaged across all 9 cross-validation runs.

7.4.2 Results

We compare our method with DT [2], and Behavior Cloning (BC). BC is implemented by (1) several convolutional layers (CNN) or (2) ViT [6] for the action prediction (imitation learning) tasks. For the position prediction task, we use the

Fig. 15. A sample frame from the collected human-following dataset. The dataset contains panoramic images, RGB and depth FOV images, and point cloud. Here, the robot is following a person (in the middle of the view) with other non-target people at the experiment site.

TABLE 1
Continuous and discrete action spaces used for offline evaluation.

Action Space	Linear Velocity × Angular Velocity
Continuous	$[-1.0, 1.0] \times [-1.0, 1.0]$
6-action	$\{0, 1.0\} \times \{-1.0, 0, 1.0\}$
10-action	$\{0, 1.0\} \times \{-1.0, -0.5, 0, 0.5, 1.0\}$

same CNN and ViT as those used in BC (labeled as CNN and ViT in Table 2).

Table 2 presents the offline evaluation results on the collected human-following dataset. Among all tasks, our method and DT, which are sequential modeling (Transformer) methods, outperform naive behavior cloning. BC-ViT generally performs worse than BC-CNN, which indicates that the sole use of ViT [6] may be insufficient to tackle robotic tasks. From the first four rows of Table 2, we find that StARformer generally produced the highest performance in all action prediction tasks. This confirms that the proposed method can be generalized to real-world imitation learning problems. With regard to the 2D position prediction task, StARformer also outperforms BC methods and DT [2] by a large margin, showing that our model can be applied to other real-world prediction tasks as well.

Moreover, our method works well with all three visual modalities, demonstrating its generalizability. The panoramic input (SRGB) provides the best performance in the continuous action prediction task, as the target object is always included in the scene, which is essential for predicting fine-grained actions. Even under FRGB input, which presents a limited view, our method outperforms the best BC agent with SRGB input. In the discrete action space setting, FRGB-D input generally yields the best performance among all three methods (except for 10-action prediction by our method). For the position prediction task, FRGB-D input yields the best general performance. Inputs with the depth channel provide more informative cues for predicting displacements than pure RGB inputs. We also find that our model with SRGB outperforms other methods with FRGB-D input in the position prediction task, without relying on any depth information. These findings based on visual modalities can be useful for future research on vision-based robot learning using Transformers.

7.5 Online Real-world Evaluation

7.5.1 Evaluation Scenarios

We designed three scenarios (see Fig. 16) based on different trajectory shapes generated by the target's motion:

(a) moving in a "0" shape (Trajectory "0")
(b) moving in an "8" shape (Trajectory "8")
(c) moving randomly with interference from non-targets (Random + Interference)

In the first two scenarios, only the target person moves and the other people remain in place. In the third scenario, non-target people also move casually in the same environment, occasionally interfering with the robot. Therefore, the difficulty increases accordingly from the first to the last scenario. Each scenario ends when the robot successfully follows the target for at least several minutes (5min, 3min, 5min for scenarios a, b, and c, respectively) or reaches a maximum of 10 trials (i.e., 10 failures). A failure is determined in real time by the experimenter when (1) the robot is about to bump into a wall, object, or other person; or (2) the robot deviates from the target person, with no indication of returning to the correct trajectory for approximate 2 seconds. We ensure fairness in the evaluation scenarios by following the same target person at the same site. The target person has an unseen appearance compared with the collected dataset. We use a panoramic image input and continuous action space in this experiment. Please check our supplementary video for a better understanding.

7.5.2 Evaluation Metrics

We devise two evaluation metrics for the human-following task.

Average Following Time to Failure (AFTF): First, we measure the Average Following Time to Failure (AFTF), or the averaged duration of each trial prior to any failure[4].

Success Rate vs. Angular Error: Second, we construct a success rate-angular error curve, showing the success rate the robot gains under a certain angular error threshold. More specifically, we define success/failure as the target person appearing within/falling out of a certain angular range (i.e., angular error) with respect to the center of the robot view (see Fig. 17. A higher success rate at a lower angular error suggests better stability in the human-following task, which means that the robot camera is continuously centered on the target. This metric is calculated using the position of the target person (center of the bounding box) extracted by a conventional tracking algorithm on panoramic images. We do not consider the distance between the robot and the target in this evaluation since our tasks do not consider a distance requirement in following.

4. The definition of a failure here in AFTF metric is the same as in Sec. 7.5.1.

TABLE 2
Offline Evaluation Results. 6-Discrete and 10-Discrete correspond to the discrete action spaces of 6 and 10 categories, respectively. We denote the evaluation metric used in each setting with ↑ / ↓ to note the preferred change. SRGB, FRGB-D, and FRGB represent for different input modalities. All listed results are are mean values among all cross-validation runs. Our method consistently outperforms the other methods in all tasks and input modalities.

Task		Metric	Method	Input		
				SRGB	FRGB-D	FRGB
Action Prediction	Continuous Action	MSE ↓	BC-CNN	0.149	0.164	0.170
			BC-ViT	0.137	0.167	0.168
			DT	0.132	0.146	0.150
			Ours	**0.124**	**0.136**	**0.142**
	6-action	Accuracy ↑	BC	58.6%	55.0%	55.4%
			BC-ViT	55.5%	49.5%	46.6%
			DT	80.6%	82.8%	82.1%
			Ours	**82.1%**	**84.0%**	**83.0%**
	10-action	Accuracy ↑	BC	39.8%	52.0%	48.2%
			BC-ViT	53.0%	46.9%	50.0%
			DT	73.6%	76.6%	75.7%
			Ours	**75.3%**	**77.0%**	**77.3%**
2-d Position Prediction		Error distance ↓	CNN	51.4	39.3	70.1
			ViT	82.5	130.5	109.9
			DT	42.9	38.7	36.8
			Ours	**32.1**	**22.8**	**23.2**

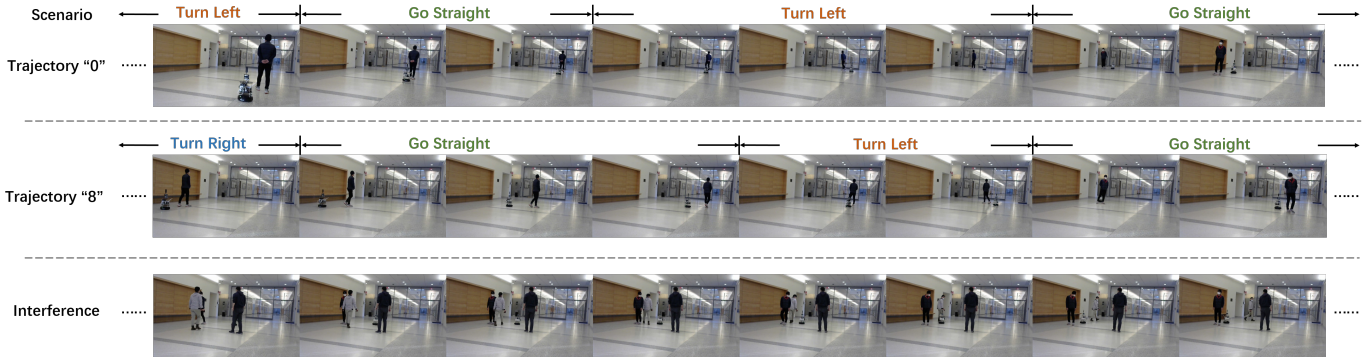


Fig. 16. Samples of the three online evaluation scenarios. We present a chronological sequence of frames from left to right for each scenario in our experiment recording (shown in different rows). The difficulty of human-following increases from Trajectory "0" (Row 1) to Trajectory "8" (Row 2) and "Random + Interference" (Row 3). In the sample with interference, we included a non-target person moving in the robot's path.

TABLE 3
Online evaluation results measured by Average Following Time to Failure (AFTF) and the number of trials (defined in Sec. 7.5.1). We skip BC-CNN in the third scenario (random + interference), as BC fails easily and the trajectories become very short. Under our method, the robot successfully follows the person in the "0" and "8" trajectories without any failure, and gains longer AFTF compared to DT and BC in the relatively-harder "Random + Interference" scenario. Please refer to our supplementary video for better understanding.

Target Trajectory	Method	trials ↓	AFTF (s) ↑
Trajectory "0"	BC	3	101.6
	DT	**1**	**300.0**
	StAR	**1**	**300.0**
Trajectory "8"	BC	10	18.1
	DT	3	62.0
	StAR	**1**	**180.0**
Random + Interference	BC	-	-
	DT	6	53.3
	StAR	**3**	**109.3**

7.5.3 Results

We acquire quantitative evaluation results (presented in Table 3 and Fig. 18) by comparing StARformer (StAR), DT [2], and behavior cloning (BC). We use BC-CNN here because BC-ViT is found to hard to generalized in this real world scenario. Table 3 lists the AFTF and the number of trials for all methods. Both Transformer-based methods outperform BC by a large margin in all scenarios. Compared with DT, our method shows better performance in the following complex patterns. In the hardest scenario, where noise and randomness make the following much more challenging, both DT and our method show shorter AFTF and a higher number of trials. However, our method is more robust, with 50% fewer trials and 105% longer AFTF than DT. We also visualize the Success Rate vs. Angular Error curve, as shown in Fig. 18. Our method shows significantly higher success rates at lower angular-error thresholds, in two of the relatively-challenging scenarios among all algorithms, demonstrating its stability when following complex trajectories.

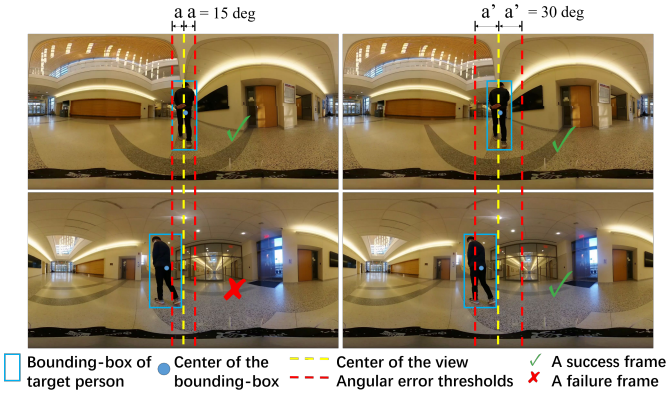


Fig. 17. Definition of success/failure in success rate vs. angular error metrics. We show two frames in different angular error thresholds (15 and 30 degrees, for example). In the left column, with a 15-deg angular error, the upper frame suggests a success because the center of the bounding-box is within the threshold, whereas the lower frame indicates a failure. By increasing the error threshold from 15 deg to 30 deg, as shown in in the right column, both frames represent success. Note that bounding-boxes of the target person are extracted after the experiment by conventional tracking algorithms, for evaluation only.

Fig 19 shows real trajectories generated from experiments on Trajectories "0" and "8". We find that for Trajectory "0" scenario, BC fails to follow the target's turning, whereas both DT and our method generate smooth trajectories. For Trajectory "8" scenario, BC fails more frequently and barely follows the target, as the trajectory includes sharper turns. In contrast, our method follows the target without any failure, whereas DT fails twice. In the "Random + Interference" scenario, we observe that our method can follow the original target for a longer time than the baseline methods. For instance, when a person crosses the target's patch, causing occlusion and confusion, the robot continues to successfully follows the target. For more details concerning the "Random + Interference" scenario, please refer to our supplementary video.

8 Conclusion

This study introduces StARformer, which explicitly models strong local relations (Step Transformer) to facilitate the long-term sequence modeling (Sequence Transformer) in Visual RL. Our extensive empirical results show how the learned StAR-representations help our model to outperform the baseline in both Atari and DMC environments, as well as both offline RL and imitation learning settings. We find that the fusion of learned StAR-representations and convolution features benefits action prediction. We further demonstrate that our designed architecture and token embeddings are essential to successfully model trajectories, with an emphasis on long sequences. We also verify that our method can be applied to real-world robot learning settings via a human-following experiment.

Acknowledgement

We thank Ryan Burgert for his contribution in setting up robot systems. We also thank members in Robotics Lab and Computer Vision Lab at Stony Brook University for valuable discussions and volunteering for data collection. This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Ministry of Science and ICT (No.2018-0-00205, Development of Core Technology of Robot Task-Intelligence for Improvement of Labor Condition.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inform. Process. Syst.*, Dec. 2017.
[2] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," in *Proc. Adv. Neural Inform. Process. Syst.*, Dec. 2021.
[3] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," in *Proc. Adv. Neural Inform. Process. Syst.*, Dec. 2021.
[4] J. Yang, C. Li, P. Zhang, X. Dai, B. Xiao, L. Yuan, and J. Gao, "Focal self-attention for local-global interactions in vision transformers," 2021, arXiv:2107.00641.
[5] H. Touvron, M. Cord, A. Sablayrolles, G. Synnaeve, and H. Jégou, "Going deeper with image transformers," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, Oct. 2021, pp. 32–42.
[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent.*, Apr. 2020.
[7] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, May 1992.
[8] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*. Citeseer, 1994, vol. 37.
[9] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, Aug. 1988.
[10] G. Tesauro *et al.*, "Temporal difference learning and TD-Gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995.
[11] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Proc. Adv. Neural Inform. Process. Syst.*, Dec. 2000.
[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013, arXiv:1312.5602.
[13] Q. Zheng, A. Zhang, and A. Grover, "Online decision transformer," 2022.
[14] H. Furuta, Y. Matsuo, and S. S. Gu, "Distributional decision transformer for hindsight information matching," in *International Conference on Learning Representations*, 2022.
[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2019, arXiv:1810.04805.
[16] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," pp. 1–12, 2018.
[17] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, 2019.
[18] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, "ViViT: A video vision transformer," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, Oct. 2021.
[19] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, P. Dhariwal, D. Luan, and I. Sutskever, "Generative pretraining from pixels," in *Proc. Int. Conf. on Mach. Learn.*, Jul. 2000, pp. 1691–1703.
[20] D. Neimark, O. Bar, M. Zohar, and D. Asselmann, "Video transformer network," 2021, arXiv:2102.00719.
[21] Y. Tang and D. Ha, "The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning," *arXiv preprint arXiv:2109.02869*, 2021.
[22] S. Dasari and A. Gupta, "Transformers for one-shot visual imitation," in *Conference on Robot Learning*, 2020.
[23] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, D. Belanger, L. Colwell, and A. Weller, "Rethinking attention with performers," in *Proc. Int. Conf. Learn. Represent.*, Apr. 2020.
[24] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," 2020, arXiv:2006.04768.

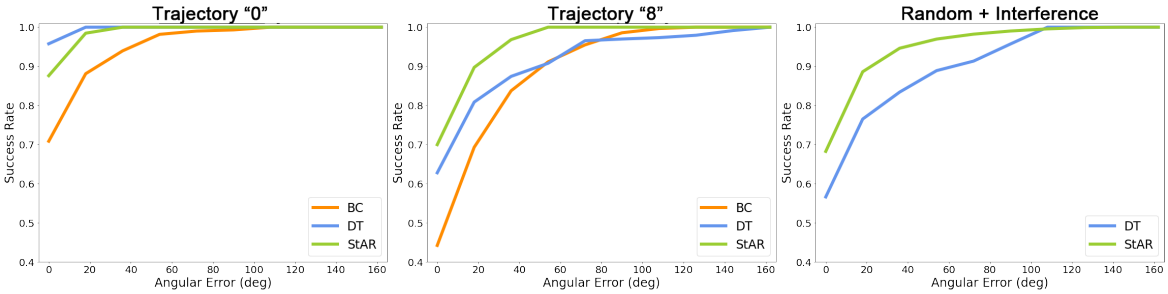


Fig. 18. Success Rate vs. Angular Error curve of online evaluation. Because BC fails easily in the third scenario, it is omitted in the figure. Our method (StAR) shows significantly higher performance compared to DT in two of the relatively-challenging scenarios (“Trajectory 8” and “Random + Interference”), thus indicating a better stability.

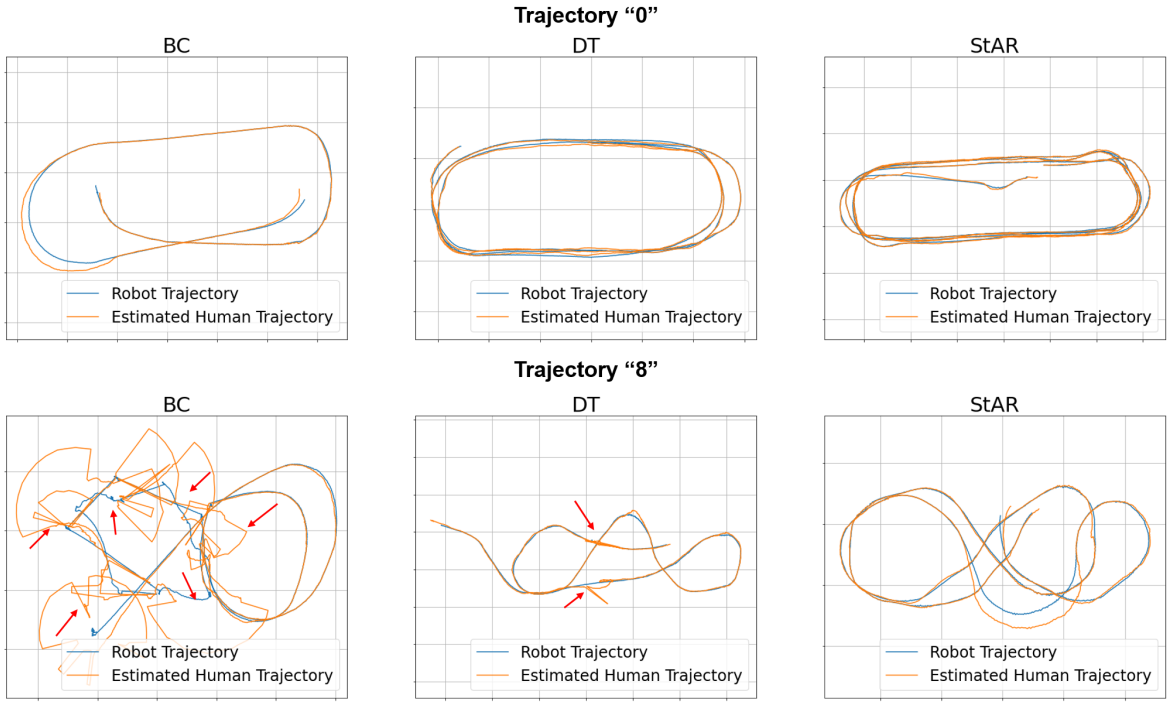


Fig. 19. Real trajectory samples (for Trajectories “0” and “8”) from online evaluation. Robot trajectories are from SLAM. Human trajectories are estimated from robot trajectories, point clouds, and images using a conventional tracking algorithm after the experiments. Red arrows indicate where the robot fails to follow the target. When the robot is successfully following the target, the estimated human trajectories are relatively consistent with the recorded robot trajectory. In the event of failure, the estimated human trajectories deviate from the robot trajectories due to (1) target being out of view or (2) our manual reset. We find our method to have the highest success rate. We present the results as movie clips in our supplementary video to improve understanding. This supplementary video also includes the results from the “random + interference” scenario, which are hard to demonstrate in static images.

[25] N. Kitaev, L. Kaiser, and A. Levskaya, “Reformer: The efficient transformer,” in *Proc. Int. Conf. Learn. Represent.*, May 2019.

[26] G. Bertasius, H. Wang, and L. Torresani, “Is space-time attention all you need for video understanding?” in *Proc. Int. Conf. on Mach. Learn.*, July 2021.

[27] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, “Swin transformer: Hierarchical vision transformer using shifted windows,” *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, pp. 10 012–10 022, Oct. 2021.

[28] Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin, and H. Hu, “Video swin transformer,” 2021, arXiv:2106.13230.

[29] M. S. Ryoo, A. Piergiovanni, A. Arnab, M. Dehghani, and A. Angelova, “TokenLearner: Adaptive space-time tokenization for videos,” in *Proc. Adv. Neural Inform. Process. Syst.*, Dec. 2021.

[30] K. Han, A. Xiao, E. Wu, J. Guo, C. Xu, and Y. Wang, “Transformer in transformer,” in *Proc. Adv. Neural Inform. Process. Syst.*, Dec. 2021.

[31] Z. Dai, H. Liu, Q. V. Le, and M. Tan, “CoAtNet: Marrying convolution and attention for all data sizes,” in *Proc. Adv. Neural Inform. Process. Syst.*, Dec. 2021.

[32] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, Jun. 2016, pp. 770–778.

[33] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” 2016, arXiv:1607.06450.

[34] Z. Lin, M. Feng, C. N. dos Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, “A structured self-attentive sentence embedding,” 2017, arXiv:1703.03130.

[35] A. P. Parikh, O. Täckström, D. Das, and J. Uszkoreit, “A decomposable attention model for natural language inference,” 2016, arXiv:1606.01933.

[36] J. Cheng, L. Dong, and M. Lapata, “Long short-term memory-networks for machine reading,” 2016, arXiv:1601.06733.

[37] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[38] S. Levine, A. Kumar, G. Tucker, and J. Fu, “Offline reinforcement learning: Tutorial, review, and perspectives on open problems,” 2020, arXiv:2005.01643.

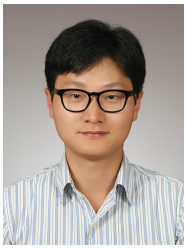
[39] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: An evaluation platform for general

agents," *J Artif Intell Res .*, vol. 47, no. 1, pp. 253–279, May 2013.
[40] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, and Y. Tassa, "dm_control: Software and tasks for continuous control," *Software Impacts*, vol. 6, p. 100022, Nov. 2020.
[41] R. Agarwal, D. Schuurmans, and M. Norouzi, "An optimistic perspective on offline reinforcement learning," in *Proc. Int. Conf. on Mach. Learn.*, July 2020, pp. 104–114.
[42] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. on Mach. Learn.*, Jul. 2018, pp. 1861–1870.
[43] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," 2021, arXiv:2004.07219.
[44] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," 2016, arXiv:1606.01540.
[45] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, "Improving sample efficiency in model-free reinforcement learning from images," in *Proc. AAAI Conf. Artif. Intell.*, May 2021, pp. 10 674–10 681.
[46] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.
[47] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.
[48] R. Agarwal, D. Schuurmans, and M. Norouzi, "An optimistic perspective on offline reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 104–114.
[49] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine, "Stabilizing off-policy q-learning via bootstrapping error reduction," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
[50] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit q-learning," 2021.
[51] S. Fujimoto and S. S. Gu, "A minimalist approach to offline reinforcement learning," in *NeurIPS*, 2021.
[52] L. P. Kaelbling, "Learning to achieve goals," in *International Joint Conference on Artificial Intelligence*, 1993.
[53] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in neural information processing systems*, 2017.
[54] V. Pong, S. Gu, M. Dalal, and S. Levine, "Temporal difference models: Model-free deep rl for model-based control," *International Conference on Learning Representations*, 2018.
[55] R. K. Srivastava, P. Shyam, F. Mutz, W. Jaśkowski, and J. Schmidhuber, "Training agents using upside-down reinforcement learning," *arXiv preprint arXiv:1912.02877*, 2019.
[56] A. Kumar, X. B. Peng, and S. Levine, "Reward-conditioned policies," *arXiv preprint arXiv:1912.13465*, 2019.
[57] A. C. Li, L. Pinto, and P. Abbeel, "Generalized hindsight for reinforcement learning," in *NeurIPS*, 2020.
[58] B. Eysenbach, X. Geng, S. Levine, and R. Salakhutdinov, "Rewriting history with inverse rl: Hindsight inference for policy improvement," in *NeurIPS*, 2020.
[59] G. Monaci, M. Aractingi, and T. Silander, "DiPCAN: Distilling privileged information for crowd-aware navigation," in *Robotics: Science and Systems (RSS) XVIII*, 2022.

Jinghuan Shang received the B.S. degree in Computer Science from Shanghai Jiao Tong University, Shanghai, China, in 2018. He is currently pursuing a Ph.D. degree in Computer Science at Stony Brook University. His research interest lies in Representation Learning, Robotics, and Computer Vision.

Xiang Li is a Ph.D. student at Computer Science Department at Stony Brook University. His research is concentrated in the area of computer vision and robotics with focus on vision-based reinforcement learning and self-supervised learning. He received B.S. in Automation and M.S. in Control Engineering (with honors) from Shanghai Jiao Tong University, Shanghai, China, in 2015 and 2018 respectively.

Kumara Kahatapitiya is a Ph.D candidate in Computer Science at Stony Brook University, broadly-interested in computer vision and machine learning with an emphasis on video understanding. He received his B.S. degree in Electronic and Telecommunication Engineering from University of Moratuwa, Sri Lanka. He is currently working on video representations with minimal supervision and attention mechanisms.



Yu-Cheol Lee received the B.S. degrees in both School of Mechanical Engineering and School of Electrical and Electronic Engineering from Yonsei University in 2004, the M.S. degree in the Department of Mechanical Engineering from POSTECH in 2006, and the Ph.D. degree in Robotics Program form KAIST in 2020. He was a Principal Researcher with Artificial Intelligence Laboratory in ETRI, Daejeon, Korea, from 2006 to 2022. He participated the visiting researcher in the Department of Computer Science, Stony Brook University, New York, US, from 2021 to 2022. Since 2022, he has joined as an assistant professor in the Department of Autonomous Vehicle Engineering, Korea Aerospace University, Goyang-si, Gyeongi-do, Korea. He has participated in numerous large-scale research project performing a leading role as a researcher and manager. His research achievements have been presented at prominent international conferences and journals including IEEE, ASME, RSJ, and KRoS, at which he received outstanding research awards. Currently, his research interests include the localization and map building for intelligent vehicles, and the navigation technology for pedestrians in indoor and outdoor environments.

Michael Ryoo is a SUNY Empire Innovation Associate Professor in the Department of Computer Science at Stony Brook University, and is also a staff research scientist at Robotics at Google. His research focuses on video representation learning. He previously was an assistant professor at Indiana University Bloomington, and was a staff researcher within the Robotics Section of NASA's Jet Propulsion Laboratory (JPL). Dr. Ryoo received his Ph.D. from the University of Texas at Austin in 2008, and B.S. from Korea Advanced Institute of Science and Technology (KAIST) in 2004. His paper on robot-centric activity recognition at ICRA 2016 received the Best Paper Award in Robot Vision.