TKN: Transformer-based Keypoint Prediction Network for Real-time Video Prediction Supplementary Materials

1st Haoran Li
University of Science and Technology of China
Hefei, China
lhr123@mail.ustc.edu.cn

2nd Pengyuan Zhou
Aarhus University
Aarhus, Denmark
pengyuan.zhou@ece.au.dk

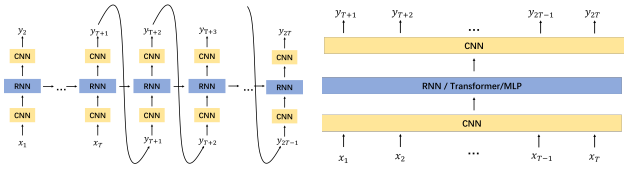
3rd Yong Liao*
University of Science and Technology of China
Hefei, China
yliao@ustc.edu.cn

I. RELATED WORKS

Unsupervised methods can reduce the cost of manual annotation which is a common requirement for video datasets.

Unsupervised keypoint learning. Due to the similarity of pixels in consecutive video frames, the keypoints in each frame can be learned via unsupervised reconstruction of the other frames. Jakab *et al.* [19] propose to learn the object landmarks via conditional image generation and representation space shaping. Minderer *et al.* [28] introduce keypoints to video prediction using stochastic dynamics learning for the first time, which drastically reduces computational complexity. Gao *et al.* [13] applied grids on top of [28] for a clearer expression of the keypoint distribution.

Unsupervised video prediction uses the pixel values of the video frames as the labels for unsupervised prediction. Existing studies can be classified into two categories, as shown by Fig. 1(a). The first category of works focuses on improving the performance of the well-known RNN by adapting the intermediate recurrent structure [4], [6], [29], [39], [40], [43], [48]. For example, E3D-LSTM [40] integrates 3DCNN with LSTM to extract short-term dependent representations and motion features. PredRNN [43] enables the cross-level communication for the learned visual dynamics by propagating the memory flow in both bottom-up and top-down orientations. The second category focuses on disentangling the dynamic objects and the static background in the video frames, mostly by adapting the Convolutional Neural Network(CNN) structure [3], [8], [15], [46], [47], [49]. For instance, DGGAN [49] trains a multi-stage generative network for prediction guided by synthetic inter-frame difference. PhyDNet [15] uses a latent space to untangle physical dynamics from residual information. The methods in both categories use so-called "sequential prediction", that is, using the previous prediction frame as the input frame for the next round of prediction. The prediction speed is proportional to the number of frames to be predicted and thus leads to an intolerably long delay for long-term prediction. Therefore, we propose a parallel prediction scheme, as shown in Fig. 1(b), to extract the features of multiple frames and output multiple predicted frames in parallel, which greatly accelerates the prediction process.



(a) Sequential prediction scheme (b) Parallel prediction scheme

Fig. 1. (a) The sequential prediction scheme generally takes a long time to predict frames due to the sequential scheme. (b) The parallel prediction scheme we propose can greatly accelerate the prediction speed.

Transformer has been utilized extensively in NLP due to its benefits over RNN in feature extraction and long-range feature capture. It monitors global attention to prevent the loss of prior knowledge which often occurs with RNN. Its parallel processing capacity can significantly accelerate the process. Recently, the field of computer vision has begun to explore its potential and produced positive results [2], [10], [17], [20], [22]–[24], [38], [45]. Most related works input segmented patches of images to the transformer to calculate inter-patch attention and obtain the features. There are also a number of vision transformer (VIT) approaches applied to video analysis [2], [12], [21], [25], [25], [26], [44]. For example, VIVIT [2] proposes four different video transformer structures to solve video classification problems using the spatio-temporal attention mechanism. [25] applies the swin transformer structure to video and uses an inductive bias of locality. In this paper we select CNN as the feature extractor instead of the VIT structure because of the huge computational cost of VIT compared to CNN. We select the transformer structure as the predictor because it outperformed RNN, mix-mlp, and other structures, in terms of predicting spatio-temporal features in our empirical experiments.

Most of the aforementioned video prediction methods extract from each frame complex features, typically of tens of thousands of bytes [1], [15], [33], [40], resulting in excessive

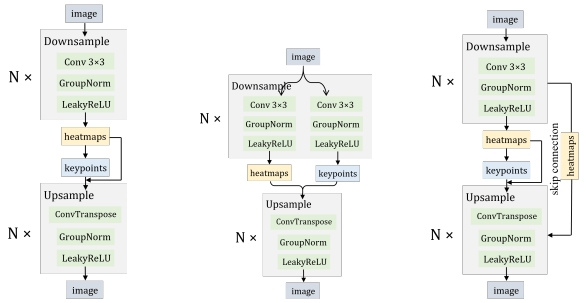


Fig. 2. Comparison of three different encoder and decoder structures. (a) The structure proposed by [28] requires more network layers while performing poorly at disentangling keypoints and background information. (b) A structure that can well disentangle keypoints and background information at the cost of complex network architecture and high computation cost. (c) We adopt the well-known skip connection to achieve good performance on information disentangling with simple structure.

numbers of floating point operations in both the feature extraction module and the prediction module. Moreover, they employ sequential (frame-by-frame) prediction process. Hence, both training and testing consume a great deal of time and memory. In the meanwhile, many videos, particularly human activity records, have a significant amount of background redundancy [18], [32] that can be removed by extracting information only from the key motions. Therefore, in this work, we try to couple the unique advantages of the transformer and the keypoint-based prediction methods to maximize their benefits.

II. MODELS

Abstract representation. Let $X, X^{'} \in \boldsymbol{X}$ denote any two frames in $\boldsymbol{X}$, and X referred as the source frame and $X^{'}$ the target frame. The keypoints in a video frame can be represented by $P = (p_1, p_2, ..., p_K) \in \Omega^K$, where K represents the number of keypoints and Ω represents the coordinates. Assume that function $\mathbb{F}$ can extract the keypoints and $\mathbb{G}$ can reconstruct the target frame $X^{'}$ by using K keypoints of $X^{'}$ and the features of the source frame X :

$$\begin{cases} \mathbb{F}(X^{'}) = P^{'} & \text{(1a)} \\ \mathbb{G}(X; P^{'}) = \hat{X}^{'}, & \text{(1b)} \end{cases}$$

where $\hat{X}^{'}$ denotes the reconstructed frame. By minimizing the difference between $\hat{X}^{'}$ and $X^{'}$, the $P^{'}$ obtained by $\mathbb{F}$ represents the different parts between X and $X^{'}$, which become what we call *keypoints*. We use the pixel-wise L_2 frame loss to measure the difference between $X^{'}$ and $\hat{X}^{'}$ as follows:

$$L_{rec} = \|X^{'} - \hat{X}^{'}\|_2. \tag{2}$$

$\mathbb{F}$ and $\mathbb{G}$ can be learned using L_{rec} in an end-to-end unsupervised learning process without labeling X or $X^{'}$.

As shown in Fig. 3(a), $\mathbb{F}$ consists of a n-layer CNN encoder E, and a coordinate generator CG which converts each heatmap output of E to $p_i^{'} = (p_{ix}^{'}, p_{iy}^{'}, p_{iv}^{'})$, where $p_i^{'}$ denotes the i-th keypoint of $X^{'}$, $(p_{ix}^{'}, p_{iy}^{'})$ represents the coordinates of $p_i^{'}$ and $p_{iv}^{'}$ denotes the intensity. $\mathbb{G}$ consists of a heatmap generator HG which converts the K keypoints to a heatmap and a n-layer CNN decoder D which has a symmetrical structure with E.

A. Keypoint Detector

Coordinate Generation (CG) module converts the heatmap generated by the encoder's last layer to the keypoints. We use a similar CG structure as in [19] which first uses a fully connected layer to convert the encoder heatmap h_n from $\mathbb{R}^{H_n \times W_n \times C_n}$ into $\mathbb{R}^{H_n \times W_n \times K}$, where K refers to the number of keypoints. We do this in the hope of compressing $H_n \times W_n$ into the form of point coordinates in the dimension of K. The converted heatmap $h_n^{'}$ can be rewritten as $h^{'}(x; y; i)$, where $x = 1, 2, ..., W_n, y = 1, 2..., H_n, i = 1, 2, ..., K$, represent the three dimensions of $h_n^{'}$, respectively. Then we can calculate the coordinates of the k-th keypiont in width p_{ix} as follows:

$$h_n^{'}(x; i) = \frac{\sum_y h_n^{'}(x; y; i)}{\sum_{x,y} h_n^{'}(x; y; i)}, \tag{3}$$

$$p_{ix} = \sum_x h_x h_n^{'}(x; i), \tag{4}$$

where h_x is a vector of length W_n consisting of values uniformly sampled from -1 to 1 (for example , if $W_n = 4$ then $h_x = [-1, -0.333, 0.333, 1]$). By doing so, we add an axis to the heatmaps at the dimension W_n where p_{ix} is the position of the i-th keypoints on the width. Similarly, we can calculate the coordinate in height p_{iy} by exchanging the position of x and y using Eq. (3) and Eq. (4). We also need the feature values at these coordinates to reconstruct the following frames with keypoints. We express such values with the averages on both the H_n and W_n dimension. We use p_{iv} to represent the value of the k-th keypoint:

$$p_{iv} = \frac{1}{H_n \times W_n} \sum_{x,y} h_n(x; y; i). \tag{5}$$

As such , we extract the keypoints as $p_i = (p_{ix}, p_{iy}, p_{iv})$,$i = 1, 2, ..., K$.

Next, the features of X and $P^{'}$ are input to $\mathbb{G}$ as shown in Eq. (1b). The features of X, $h_n \in \mathbb{R}^{H_n \times W_n \times C_n}$, are obtained via E. $P^{'}$ is converted to a heatmap $h_p^{'}$ via HG :

$$HG(p_1^{'}, ..., p_i^{'}, ..., p_K^{'}) = h_p^{'}. \tag{6}$$

Heatmap Generation (HG) module is a reversed process of CG that converts coordinates to the heatmap. We use a 2-D Gaussian distribution to reconstruct the heatmaps. We first convert the coordinates p_x, p_y into 1-D Gaussian vectors, x_{vec} and y_{vec}, where $p_x = (p_{1x}, p_{2x}, ..., p_{Kx}), p_y = (p_{1y}, p_{2y}, ..., p_{Ky})$, as follows:

$$x_{vec} = exp(-\frac{1}{2\sigma_2} \|p_x - \bar{p}_x\|^2), \tag{7}$$

$$y_{vec} = exp(-\frac{1}{2\sigma_2} \|p_y - \bar{p}_y\|^2), \tag{8}$$

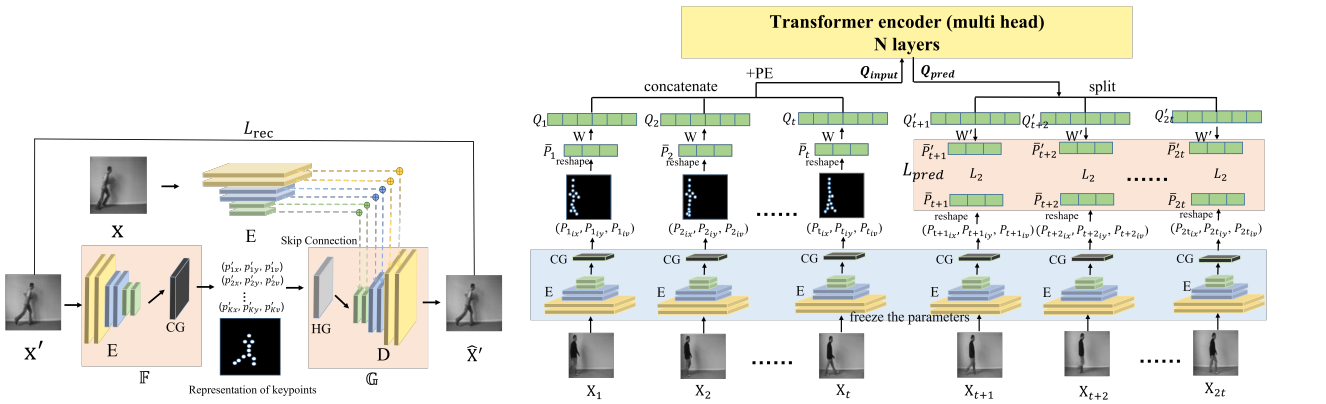


Fig. 3. Detailed structure of TKN. Two main modules are the Keypoint Detector and the Predictor marked with the blue and yellow undertones. The predicted frame uses the background information extracted from the last frame of the input. Both the inputting stage and prediction stage allow batch processing (e.g., input multiple frames simultaneously) and thus enable temporal parallelism. Note that the ground truth keypoints information, $P_{real} = (\bar{P}_{t+1}, ..., \bar{P}_{2t})$, is output by $X_{t+1}, ..., X_{2t}$ using keypoint detector (excluded from the figure for simplicity). L_{pred} loss is marked with the red undertones.

where $\bar{p}_x$ and $\bar{p}_y$ are the expectations of p_x and p_y, respectively. By multiplying x_{vec} and y_{vec} we can get the 2-D Gaussian maps G_maps as follows:

$$G\_maps = x_{vec} \times y_{vec}. \tag{9}$$

Finally we calculate the Hadamard product of G_maps and p_v to get the $h_p^{'}$:

$$h_p^{'} = G\_maps \circ p_v. \tag{10}$$

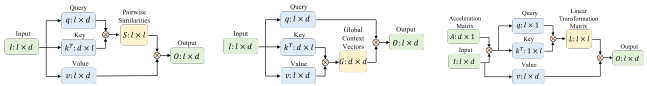
We align the dimension of $h_p^{'}$ with h_n to allow their direct concatenation sent to the decoder for reconstruction. As mentioned, inspired by the "skip connetion" in UNet [31] and Ladder Net [30], which can reconstruct images better with fewer encoder and decoder layers, we input the heatmaps $h_1, h_2, ..h_n$ obtained by each encoder layer to the decoder through "skip connection". Let D_i denote the i-th decoder layer and d_i denote its output heatmap, where $d_1 = D_1(concat(h_p^{'}, h_n))$ and each subsequent layer can be expressed as follows:

$$d_i = D_i(concat(d_{i-1}, h_{n-i+1})), i \in \{2, n\}, \tag{11}$$

where d_n is $\hat{X}'$. In this manner, the decoder learns the "background" (i.e., the static information) features eliminated by the encoder, thus improving the higher level representation details of the model. The additional "background" information also allows the encoder to focus more on the keypoints.

B. Predictor

The original prediction task is transformed, via the keypoint detector's encoding, into predicting the subsequent m groups of keypoints $P_{t+1}, ..., P_{t+m}$, based on the prior n groups of keypoints $P_{t-n+1}, ..., P_t$. We select Transformer as the predictor due to the benefits explained in the Related Work. Our experiments revealed that utilizing only the transformer's encoder to encode the temporal relationship between keypoints yields better prediction results to using the transformer's entire structure, and at a faster rate. Therefore, we opted to use only the encoder part.



(a) Dot-Product Attention (b) Linear Attention (c) Ours

Fig. 4. Three different attention mechanism structures: (a) is the original dot-product structure, with a computational complexity of $O(l^2 d)$; (b) is the linear attention structure, with a computational complexity of $O(ld^2)$); (c) is the structure we propose, which uses an acceleration matrix A to reduce the computational complexity of the linear transformation matrix L to $O(ld+l^2)$.

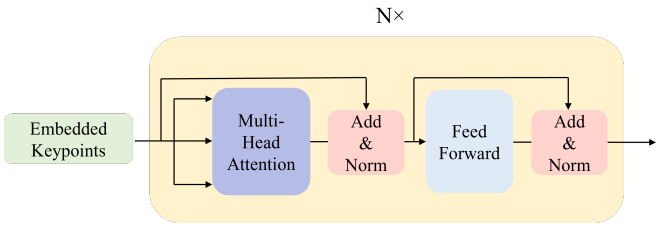


Fig. 5. The structure of transformer encoder. Embedded keypoints are the explicit keypoints coordinates mapping to the high-dimensional implicit representations combined with temporal position encoding.

Transformer encoder.

We use the transformer encoder structure from [37] as shown in Fig. 5. The transformer encoder uses a self-attention mechanism that calculates the relationship between the current and other inputs. In this paper, we use the transformer encoder to calculate how the keypoints of the current frame are composed of the previous keypoints.

Self-attention is a function that maps a query (q) and a set of key-value (k,v) pairs to an output. Each input corresponds to a vector query of length d_q, a vector key of length d_k, and a vector value of length d_v. Suppose we have l inputs denoted

as $I \in \mathbb{R}^{l \times d_{model}}$, each of which has a dimension of d_{model} as mentioned in the paper, there are l query, key, and value vectors dentoed as $q \in \mathbb{R}^{l \times d_k}$, $k \in \mathbb{R}^{l \times d_k}$, and $v \in \mathbb{R}^{l \times d_v}$, respectively. We use the fully connected layer to convert the inputs to q, k and v:

$$\begin{cases} q = IW^q \\ k = IW^k \\ v = IW^v, \end{cases} \tag{12}$$

where $W^q \in \mathbb{R}^{d_{model} \times d_k}$, $W^k \in \mathbb{R}^{d_{model} \times d_k}$, and $W^v \in \mathbb{R}^{d_{model} \times d_v}$ are the conversion matrices. The general self-attention structure, as shown in Fig. 4(a), is known as dot-product attention, and can be expressed as follows:

$$Attention(q, k, v) = softmax(\frac{qk^T}{\sqrt{d_k}})v, \tag{13}$$

where $\sqrt{d_k}$ is used to prevent the gradient disappearance when the q and k dot product is too big.

We employ this attention mechanism to represent the predicted l keypoints sequence as a linear transformation of the input l keypoints sequence. We use v to represent the value of the keypoints, and qk^T to represent the coefficients of the linear transformation matrix. Therefore, the attention mechanism in Eq. (13) can be expressed as follows:

$$\begin{pmatrix} v_1^{'} \\ \vdots \\ v_l^{'} \end{pmatrix} = \begin{pmatrix} \omega_{11} & \cdots & \omega_{1l} \\ \vdots & \ddots & \vdots \\ \omega_{l1} & \cdots & \omega_{ll} \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ v_l \end{pmatrix}, \omega_{ij} = \frac{1}{\sqrt{d_k}} \sum_a (q)_{ia} (k^T)_{aj}, \tag{14}$$

where ω_{ij} in the qk^T matrix represents the coefficient of the i-th predicted value $v_i^{'}$ on v_j. The computational complexity of such attention is $O(l^2 d)$(When calculating complexity, we simply assume that $d_k = d_v = d$), which is significantly high. This differs from many methods that reduce attention's computational complexity. For example, linear attention, which is based on the assumption in natural language processing where $l >> d$, computes the kv matrix first and then the product with q, resulting in a complexity of $O(ld^2)$. Its structure is shown in Fig. 4(b). However, in the field of video prediction, it is often the case that $l < d$, hence such improvements would increase complexity. Therefore, as shown in Fig. 4(c), we introduce an acceleration matrix $A \in \mathbb{R}^{d_k \times 1}$. After multiplying it with the input I, we obtain q_A and k_A matrices with reduced dimensions. Then we compute the linear transformation matrix $L = q_A k_A^T$ to reduce the computational complexity of the linear transformation matrix to $O(ld + l^2)$. The overall complexity is thus reduced by half. Note that though it's also possible to reduce the dimensions of the v matrix, it would significantly lower the model prediction accuracy. Therefore, we only apply this operation to the qk_T matrix, which barely affects the accuracy.

In fact, we use multi-head self-attention mechanisms for better performance with more parameters. Each head refers to self-attention in Eq. (13). We concatenate all the heads together and pass them through a fully connected layer to get the result of the same dimension with the input I,

$$Multihead(q, k, v) = concat(head_1, ..., head_n)W^h, \tag{15}$$

where $head_i = Attention(IW_i^q A, IW_i^k A, IW_i^k A), i = 1, 2, ..., n$, $W^h \in \mathbb{R}^{nd_k \times d_{model}}$. We add the result to the input I by residual connection and then normalize them by LayerNorm to solve the gradient disappearance:

$$x = LayerNorm(I + Multihead(q, k, v)). \tag{16}$$

Considering that the self-attention mechanism may not fit complex processes well, the model is enhanced by adding two fully connected layers with a ReLU activation in between called the Feed-Forward Network:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2, \tag{17}$$

Then we use Eq. (16) again to get the final result. We can either output the result directly or send it as input to the Transformer encoder for multiple iterations.

The transformer encoder requires the input to be a one-dimensional vector with a length of d_{model}, e.g., $(512, 768, 1024)$. Hence we first convert the K keypoint-triples $\{(p_{ix}, p_{iy}, p_{iv}) | i = 1, .., K\}$ to a one-dimensional vector $\bar{P}$:

$$P = \begin{cases} p_1 = (p_{1x}, p_{1y}, p_{1v}) \\ p_2 = (p_{2x}, p_{2y}, p_{2v}) \\ \vdots \\ p_K = (p_{Kx}, p_{Ky}, p_{Kv}) \end{cases}$$

$$\downarrow$$

$$\bar{P} = (p_{1x}, p_{1y}, p_{1v}, ..., p_{Kx}, p_{Ky}, p_{Kv}). \tag{18}$$

$\bar{P}$ represents the low-dimensional *explicit* spatial coordinates. Inputting $\bar{P}$ directly into the transformer necessitates adjusting the dimension of the intermediate parameter values according to K in each prediction instance. Consequently, training and testing would become difficult. As mentioned, the predicted l keypoints sequence is obtained through the linear transformation of the input keypoints sequence. However, in real-world scenarios, the motion of objects is difficult to represent solely with linear equations; it generally requires the use of differential equations. On the other hand, if we transform the explicit input into a higher-dimensional latent space, the features of the keypoints sequence can be considered predictable through linear transformation. Moreover, many works [5], [14], [27], [35], [50], [51] have demonstrated the benefits of *latent* representations. We also found latent representations well capture the regularity of keypoints over time in our experiments. Therefore, we use a matrix to map the explicit coordinates representation to a latent space to obtain a high-dimensional latent representation vector. Specifically, we convert the variable-length and low-dimensional $\bar{P}$ to a fixed-length and high-dimensional Q by converting $\mathbb{R}^{3K}$ to $\mathbb{R}^{d_{model}}$ via a mapping matrix W, where $W \in \mathbb{R}^{d_{model} \times 3K}$, as follows:

$$Q = W \cdot \bar{P}. \tag{19}$$

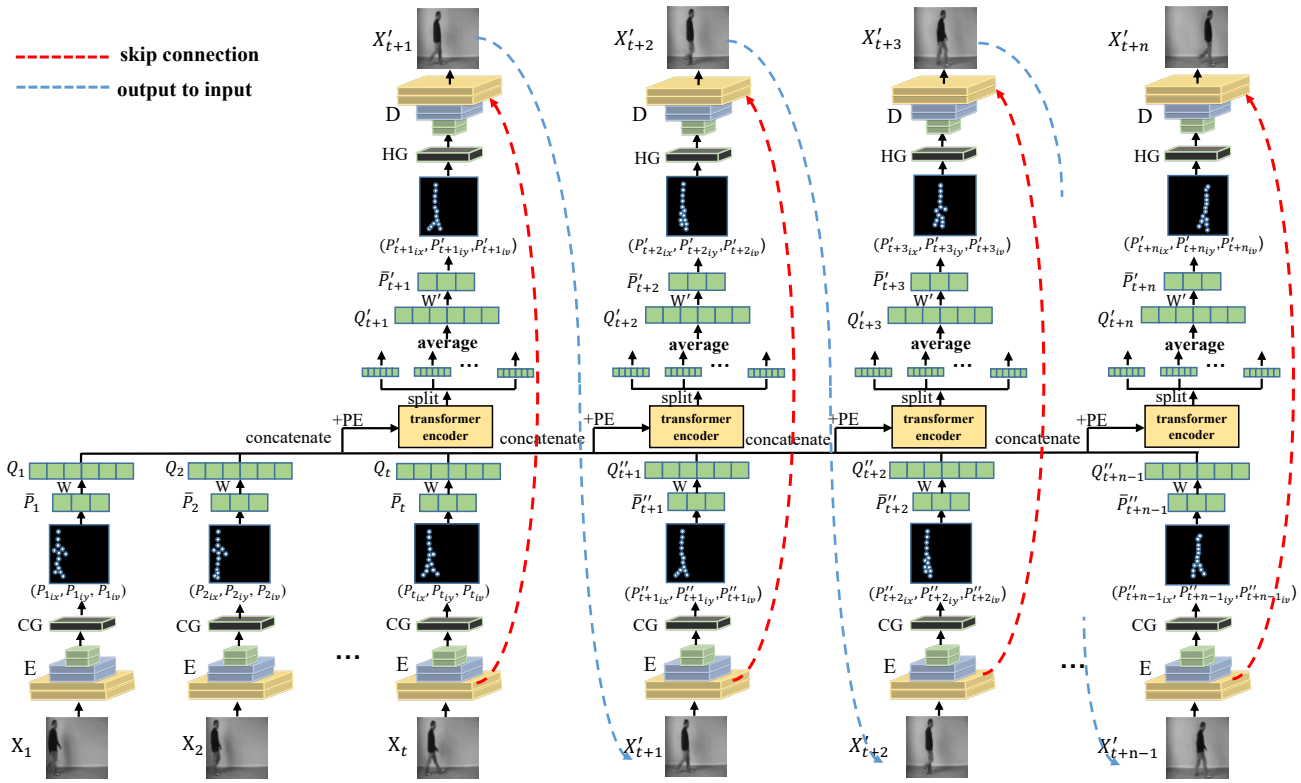
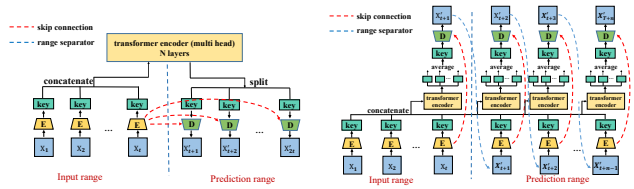


Fig. 6. Detailed structure of TKN-Sequential. It uses the same keypoint detector and predictor structures with TKN but has a different prediction process. Particularly, it uses the previous predicted frame's background as the following one's to ensure background consistency.



(a) TKN (b) TKN-Sequential

Fig. 7. Structure comparison of TKN and TKN-Sequential. (a) TKN is a parallel prediction scheme that inputs a batch of frames simultaneously and predicts a batch of frames simultaneously. It uses one transformer encoder to predict t keypoints via the background reduction of the t-th frame. (b)TKN-Sequential is a sequential prediction scheme that makes prediction frame by frame. It uses one transformer encoder per predicted frame via the background information from each previous frame.

To compensate the lack of time-sensitive capability, we manually add location information position embedding (PE) to Q,

$$Q_{input} = concat(Q_{t-n+1}, ..., Q_t) + PE, \tag{20}$$

where PE is the trigonometric function as defined in [37]:

$$\begin{cases} PE_{pos,2i} = \sin \dfrac{pos}{10000^{\frac{2i}{d_{model}}}} \\ PE_{pos,2i+1} = \cos \dfrac{pos}{10000^{\frac{2i}{d_{model}}}}, \end{cases} \tag{21}$$

where pos and i represent the sequential order of the input and the i-th element of the input. The number of input sequences and output sequences are equal for the transformer encoder. Therefore, we can use transformer encoder to get the predictions $Q^{'}_{t+1}, ..., Q^{'}_{2t}$ using the input Q_{input}:

$$Q^{'}_{t+1}, ..., Q^{'}_{2t} = Trans\_encoder(Q_{input}). \tag{22}$$

Finally, we use an invert mapping matrix $W^{'} \in \mathbb{R}^{3K \times d_{model}}$ to reconstruct the high-dimensional sequence $Q_{pred} = (Q^{'}_{t+1}, ..., Q^{'}_{2t})$ back to the low-dimensional keypoints spatio-temporal sequence P_{pred}, which is then input to the decoder to generate the predicted frames:

$$P_{pred} = W^{'} \cdot Q_{pred}. \tag{23}$$

We only need to calculate the loss of sequence $P_{real} = (\bar{P}_{t+1}, ..., \bar{P}_{2t})$ which is output by $X_{t+1}, ..., X_{2t}$ using keypoint detector and P_{pred} to complete the training of the predictor using a well-trained keypoint detector, for which we also use the L_2 loss:

$$L_{pred} = \|P_{real} - P_{pred}\|_2. \tag{24}$$

C. Prediction Processes

Sequential prediction is time consuming. Since most subsequent frames in high frame-rate videos are fairly similar, we can use the background of the frame immediately just before the prediction target as the background of the prediction target frame. We can then combine the predicted keypoints with the background to generate the integrated predicted

frames. As illustrated in Fig. 3(b), we integrate the P_{pred} generated by Transformer encoder to the background of t-th frame and directly generate the subsequent t prediction frames $X^{'}_{t+1}, X^{'}_{t+2}, ..., X^{'}_{2t}$. This parallel prediction mechanism, i.e., TKN, can input and predict multiple frames as batches to significantly accelerate the prediction process.

We reason that frame-by-frame prediction structure has higher accuracy to predict frames with frequent changes (as proved by experiment results). Hence we also provide a sequential variation of TKN, TKN-Sequential, which uses the previous predicted frame's background as the following one's to ensure background consistency. Fig. 6 shows the detailed structure of TKN-Sequential and Fig. 7 depicts its comparison with TKN.

Since the output of the transformer encoder has the same length as the input sequence, we take the averaged predictor's output as the predicted frame. Suppose we have predicted i frames, $Q^{'}_{t+1}, ..., Q^{'}_{t+i}$, then $Q^{'}_{t+i+1}$ can be expressed as:

$$Q^{'}_{t+i+1} = \frac{1}{t+i} \sum_{j=1}^{t+i} Trans\_encoder(Q_{input}; j), \qquad (25)$$

where $Q_{input} = concat(Q_{t-n+1}, ..., Q_t, Q^{'}_{t+1}, ..., Q^{'}_{t+i}) + PE$ according to Eq. (20).

$X^{'}_{t+1}$ is the combination of predicted $\bar{P}^{'}_{t+1}$ and the background information of X_t extracted by the decoder. Note that "$\bar{P}^{''}_{t+1}$ and the background information of $X^{'}_{t+1}$" are not equal to "$\bar{P}^{'}_{t+1}$ and the background information of X_t", albeit they are both extracted by the encoder from the $X^{'}_{t+1}$. It is because two consecutive frames are very similar but still have some minor differences in the keypoints and background information, otherwise it would be no different from multi-frame prediction process.

III. Experimental Setup

Dataset. We used two real action datasets, KTH [32] and Human3.6 [18], to verify the real-time and efficient performance of the proposal under different patterns. We also conducted tests on the Moving MNIST dataset [34] for non-human motion videos and the Caltech Pedestrian dataset [9] for complex driving videos, to validate the superiority of our method across different types of videos.

Evaluation metrics. Traditional evaluation metrics include structural similarity (SSIM) and peak signal-to-noise ratio (PSNR). Higher SSIM indicates a higher similarity between the predicted image and the real image. Higher PSNR indicates better quality of the reconstructed image. We also quantify the resource (time and memory) consumption, for which a uniform batch size of 32 and 1 are used for KTH dataset during training and testing, and 16 and 1 are used for Human3.6 dataset during training and testing.

In addition, we measure the FLOPs (floating-point operations per second) to assess the computational cost and the number of parameters of the model with the thop[1] package.

[1]https://pypi.org/project/thop/

Baselines. To validate the real-time performance of TKN, we select 8 most classical and effective SOTA methods ConvLSTM [33], Struct-VRNN [28], Grid-Keypoint [13], PredRNN [41], PredRNNv2 [42], PhyDNet [15], E3D-LSTM [40] and SLAMP [1]as the baselines, all of which are implemented with Pytorch for fair comparisons.

IV. Results and Analysis

A. Speed and Accuracy

Table I summarizes the performance comparison on KTH and Human3.6 datasets. For KTH, we input 10 frames to predict 10 frames during training and 20 frames during testing. For Human3.6, we input 4 frames to predict 4 frames during both training and testing. *TIME (train)* refers to the average period length per training epoch in seconds. *Time (test)* indicates the period length from inputting the frames to after generating the predicted frames in milliseconds. *FPS* is the number of generated frames per second calculated via *Time (test)*. *Memory* indicates the maximum memory consumption at a stable status. Note that to ensure fair comparisons with the end-to-end training methods, *TIME (train)* and *Memory (train)* of TKN, Struct-VRNN and Grid-Keypoint in Table I are all tested without freezing parameters. The results show that TKN outperforms most baselines in both speed and memory consumption significantly with only minor accuracy deterioration on both datasets.

KTH results show that TKN performed 19 times faster than the best method E3D-LSTM with only 0.9% and 5.5% degradation in SSIM and PSNR during testing, while reducing memory consumption by at least 12.7% (training) and 0.9% (testing) compared to the second best methods Struct-VRNN and PredRNN. As such, TKN can bear up to as large a batch size as 150 with up to 24 GB memory which no baseline can even come close to. TKN is 4 times faster than TKN (w/o tp). Fig. 8 shows the performance of long-range prediction performance tested on the walking class of KTH, with 10 frames as input for predicting 40 frames. The result shows that the TKN predicts the position and pose of a person fairly well while TKN-Sequential presents more and clearer details, because TKN only uses the background information of a fixed frame to synthesize the following frames.

We compare the performance of our models on the different action classes contained in KTH, each class with 100 randomly selected video sequences of each KTH's action class for tests. As summarized in Table II, TKN-Sequential performs better than TKN on actions with large movements such as walking, jogging, and running, while TKN performs better on handwaving, handclapping, and boxing which have smaller movements.

Human3.6 results show that TKN outperforms the baselines on accuracy performance. Moreover, TKN reduces time and memory consumption by 6% and 49% during training, and 66% and 9% during testing, compared to the second best alternative. Fig. 9 depicts the comparison of TKN and baselines

TABLE I

THE RESULTS ON KTH AND HUAMN3.6. ↑ MEANS THE HIGHER THE BETTER AND ↓ MEANS THE LESS THE BETTER. WE SKIPPED SOME TESTS DUE TO THE LACK OF ORIGINAL CODE. INSTEAD, WE USED THE RESULTS PROVIDED BY THE ORIGINAL PAPERS (INDICATED BY "*"), OR SKIPPED IF THE PAPERS DIDN'T PROVIDE RESULTS (INDICATED BY "-"). "W/O TP" MEANS WITHOUT TEMPORAL PARALLEL AND "SEQ" MEANS SEQUENTIAL. WE USED 32 AND 16 AS THE DEFAULT BATCH SIZES FOR KTH AND HUMAN3.6, BUT 16 AND 8 FOR A FEW EXCEPTIONS WHICH OTHERWISE EXCEEDED THE GPU CAPACITY DUE TO TOO MANY INTERMEDIATE RESULTS GENERATED BY THE ALGORITHMS LIKE SLAMP AND E3D-LSTM. (30) INDICATES USING 10 INPUT FRAMES TO PREDICT 30 FRAMES WITH SLAMP. **STRUCT-VRNN** AND **GRID-KEYPOINT** ARE KEYPOINT-BASED BASELINES.

Dataset	Method	SSIM↑	PSNR ↑	TIME (s)↓ (train)	TIME (ms)↓ (test)	FPS↑ (test)	Memory (MB)↓ (train)	Memory (MB)↓ (test)
KTH	ConvLSTM	0.712*	23.58*	61	72	278	8,055	1,779
	PredRNN	0.839*	27.55*	204	184	109	6,477	1,721
	PredRNNv2	0.838*	28.37*	246	222	90	8,307	1,779
	PhyDNet	0.854	26.9	108	240	83	8,491	2,704
	SLAMP	0.864*(30)	28.72(30)	465	388	52	21,103(16)	2,295
	E3D-LSTM	**0.879***	**29.31***	879	338	59	21,723(16)	2,687
	Grid-Keypoint	0.837*	27.11*	145	252	79	12,661	2,259
	Struct-VRNN	0.766*	24.29*	111	151	132	5,661	1,817
	TKN	0.871	27.71	**35**	**17**	**1,176**	4,945	1,705
Human3.6	ConvLSTM	0.776*	-	63	32	125	6,561	1,857
	PredRNN	0.781*	-	462	47	85	5,829	1,743
	E3D-LSTM	0.869*	-	3154	167	24	18,819(8)	5,767
	PhyDNet	0.901*	-	207	88	45	12,213	2,353
	Grid-Keypoint	0.928	28.76	114	106	38	9,891	2,003
	Struct-VRNN	0.916	26.97	67	41	98	5,015	1,962
	TKN	**0.958**	**30.89**	64	**11**	**364**	2,561	1,587

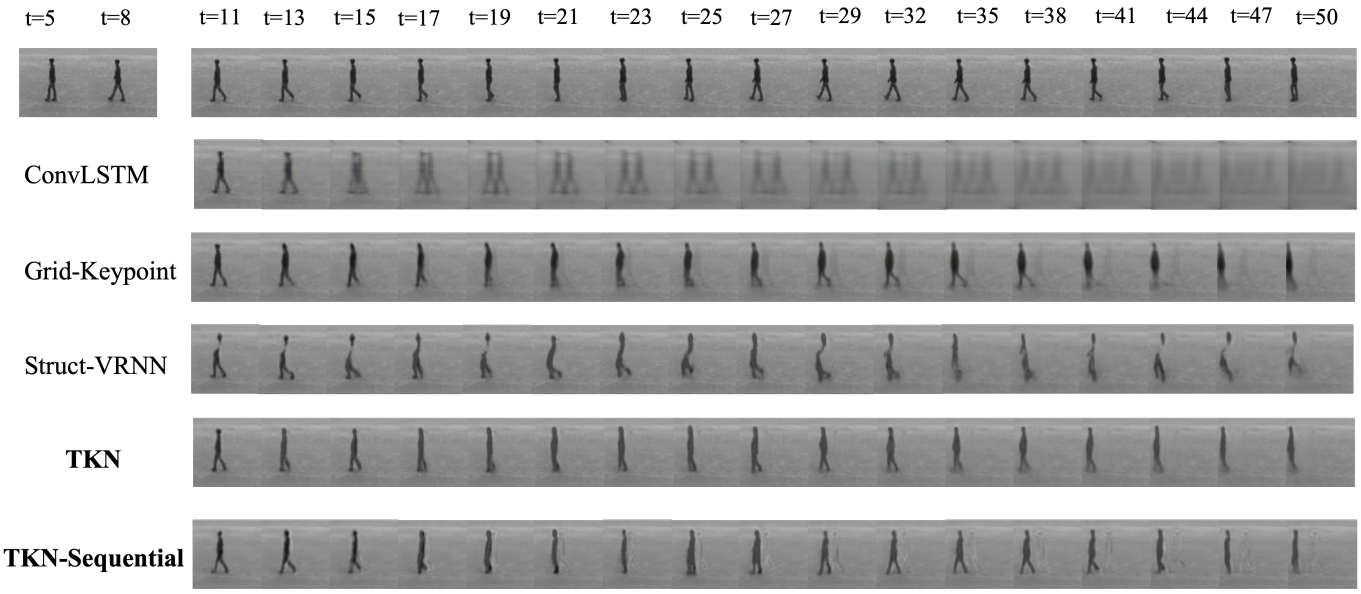


Fig. 8. Results of long-range predictions on KTH. TKN and TKN-Sequential perform better than the baselines. TKN-Sequential provides more precise details. ($10 \rightarrow 20$)

on the Human3.6 dataset for short-range prediction, as most baselines do. The changes in the actions are small within a short period. But upon closer observation, we can tell that the lighting of the background and the movement of the person in TKN are closest to the goundtruth.

Moving Mnist and Caltech Pedestrian. We did not test the prediction speed on these two datasets since it's only related to the image shape. As shown in Fig. **??** and Fig. **??**, TKN predicted the outcomes with good accuracy on both Moving Mnist and Caltech Pedestrian datasets. As summarized in Tables IV and V, TKN achieved comparable performances to the state-of-the-art (SOTA).

Deeper speed analysis. To help understand why TKN runs so much faster than SOTA algorithms, we measure FLOPs and the number of parameters for each method. TKN and TKN(w/o tp) share the same structure and thus have the same numbers of FLOPs and parameters. As shown in Table III, TKN and TKN-Sequential have much fewer FLOPs than the baselines, indicating their much higher computation efficiencies. Table VI summarizes the detailed comparison

TABLE II
SSIM PERFORMANCES ON DIFFERENT KTH'S ACTIONS.

Method	boxing	handclapping	handwaving
TKN	0.897	0.908	0.898
TKN-Sequential	0.878	0.874	0.857
Method	jogging	running	walking
TKN	0.762	0.759	0.806
TKN-Sequential	0.783	0.775	0.820

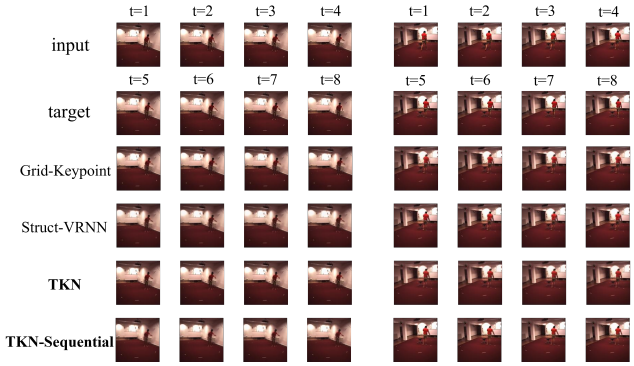


Fig. 9. Results of short-range predictions on Human3.6. (4 → 4)

TABLE III
THE RESULTS OF FLOPS AND THE NUMBER OF PARAMETERS.

Method	FLOPs (G) ↓	Params (M)↓
ConvLSTM	93.7	4.8
PredRNN	29.4	6.0
PredRNNv2	29.6	6.1
PhyDNet	21.7	**3.0**
SLAMP	95.0	49.4
E3D-LSTM	270.2	3.7
GridNet	26.2	3.3
Struct-VRNN	11.8	4.3
TKN	**1.6**	19.1
TKN (w/o tp)	**1.6**	19.1
TKN-Sequential	3.5	105

between TKN, TKN(w/o tp), and the other two Keypoint-based methods. For the choice of predictor, TKN uses transformer encoder while Grid-Keypoint uses convlstm and Struct-VRNN uses VRNN. The results in Table VI show that TKN presents the fastest speed in both modules and provides an

TABLE IV
QUANTITATIVE RESULTS ON THE MOVING MNIST DATASET.

Method	MSE ↓	MAE↓	SSIM↑
ConvLSTM	103.3	182.9	0.707
PredRNN	56.8	126.1	0.867
PredRNNv2	48.4	-	0.891
PhyDNet	24.4	70.3	0.947
MIM	44.2	101.1	0.910
E3D-LSTM	41.3	86.4	0.910
CrevNet + ConvLSTM	38.5	-	0.928
CrevNet + ST-LSTM	**22.3**	-	**0.949**
SimVP	23.8	-	0.948
TKN	24.1	70.1	0.948
TKN-Sequential	24.7	72.9	0.945

TABLE V
QUANTITATIVE RESULTS ON THE MOVING MNIST DATASET.

Method	SSIM↑	PSNR ↑
BeyondMSE	0.847	-
MCnet	0.879	-
PredNet	0.905	27.6
ContextVP	0.921	28.7
rCycleGan	0.919	29.2
CrevNet	0.925	29.3
STMFANet	0.927	29.1
SimVP	**0.940**	**33.1**
TKN	0.925	29.6
TKN-Sequential	0.927	29.7

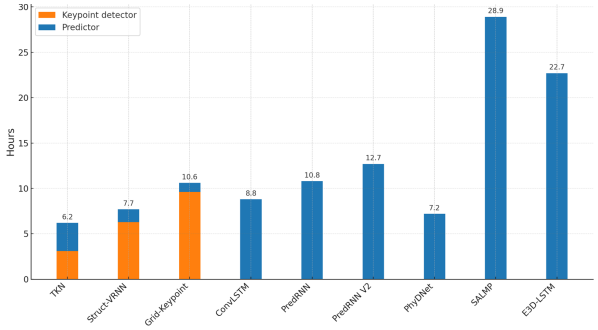


Fig. 10. Total training time of TKNand baselines.

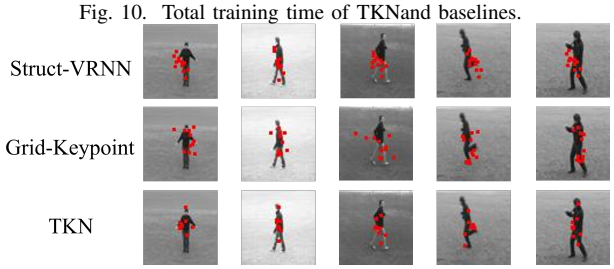


Fig. 11. Comparison of the keypoints extracted by different methods

8 times speedup compared with TKN(w/o tp), indicating the key role of keypoint detector in terms of prediction speed and the advantage of the parallel scheme. We can also find that although the predictor of TKN has more FLOPs and parameters due to the larger number of parameters of the employed transformer encoder [37], it runs about 5 to 10 times faster than the others.

Overall training time. We compare the overall training time considering various numbers of required epochs before convergence. Note that here TKN, Grid-Keypoint, and Struct-VRNN use the two-step training as mentioned in Section III.

TABLE VI
TIME, FLOPS, AND THE NUMBER OF PARAMETERS COMPARISONS BETWEEN THE KEYPOINT-BASED MODELS.

Method	Keypoint detector			Predictor		
	Time (ms)	FLOPs (G)	Params (M)	Time (ms)	FLOPs (G)	Params (M)
Struct-VRNN	104	11.8	0.8	38	**0.1**	3.5
Grid-Keypoint	142	17.7	1.8	84	8.5	**1.7**
TKN (w/o tp)	67	**1.4**	**0.1**	**8.3**	0.2	18.9
TKN	**8.2**	**1.4**	**0.1**	**8.3**	0.2	18.9

TABLE VII
THE INFLUENCE OF CONVOLUTION KERNEL SIZE ON THE RECONSTRUCTION ACCURACY AND INFERENCE SPEED OF KEYPOINT DETECTOR. "3×3, 1×1" INDICATES USING 1×1 CONVOLUTION KERNEL FOR LAYERS THAT CHANGE ONLY THE CHANNEL SIZE AND NOT THE HEATMAP SIZE AND 3×3 CONVOLUTION KERNEL FOR THE OTHER LAYERS.

Conv Kernel	SSIM	PSNR	Speed(ms)	FLOPs(G)	Params(M)
3×3	**0.916**	**31.91**	8.2	1.4	0.14
2×2	0.862	28.55	7.6	0.5	0.06
1×1	0.851	28.07	**7.1**	**0.2**	**0.02**
3×3, 1×1	0.900	30.84	7.8	0.9	0.08

TABLE VIII
THE RESULTS OF FRAME RECONSTRUCTION AND PREDICTION USING DIFFERENT NUMBERS OF KEYPOINTS. "SEPARATION-NET" REFERS TO THE STRUCTURE IN FIG. 2(B). 12, 12, 16, ARE THE NUMBER OF KEYPOINTS WHEN STRUCT-VRNN, GRID-KEYPOINT, AND SEPARATION-NET ACHIEVED THEIR BEST PERFORMANCES, RESPECTIVELY.

Method	Num	Reconstruction		Prediction	
		SSIM	PSNR	SSIM	PSNR
Struct-VRNN	12	0.821*	27.86*	0.766*	24.29*
Grid-Keypoint	12	0.862*	29.68*	0.837*	27.11*
Separation-Net	16	0.900	30.95	0.855	26.81
TKN	4	0.895	30.43	0.850	26.66
	8	0.909	31.28	0.854	26.72
	12	0.914	31.71	0.863	27.40
	16	0.916	31.91	**0.871**	**27.71**
	20	0.915	31.95	0.861	27.23
	32	0.922	32.48	0.858	26.92
	64	**0.925**	**32.71**	0.849	26.46

TABLE IX
PERFORMANCE COMPARISON BETWEEN TKN EMPLOYING DIFFERENT PREDICTORS. 'DOT-PRODUCT' REFERS TO DOT-PRODUCT ATTENTION. 'LINEAR' REFERS TO LINEAR ATTENTION.

Method	SSIM	PSNR	TIME (s) train	TIME (ms) test	FPS test	Memory (MB) train	Memory (MB) test	FLOPs(G)	Params(M)
TKN	**0.871**	**27.71**	**13**	17	1,176	4,945	**1,705**	1.6	19.1
TKN+Dot-Product	0.871	27.70	14	18	1,111	4,975	1,787	1.7	19.0
TKN+Linear	0.869	27.68	15	20	1,000	5,067	1,851	1.9	19.0
TKN+RNN	0.826	25.61	**13**	15	1,333	5,479	2,131	1.9	47.5
TKN+LSTM	0.815	25.19	19	23	870	9,343	4,287	3.3	189.1
TKN+GRU	0.829	25.82	17	22	1,000	8,029	3,567	2.8	141.9
TKN+MLP	0.814	25.12	**13**	**13**	**1,538**	**4,841**	1,733	**1.5**	**14.4**

TABLE X
PERFORMANCE COMPARISON OF TKN'S PREDICTION MODULE BETWEEN USING ONLY THE TRANSFORMER ENCODER AND WHOLE TRANSFORMER.

Method	SSIM	PSNR	TIME (ms) all model(test)	FPS (test)	Memory (MB) (test)	TIMES (ms) Predictor(test)
TKN (employs only the encoder)	**0.871**	**27.71**	**17**	**1,176**	**1,705**	**8.3**
TKN (employs whole transformer)	0.800	25.87	93	215	1,759	74

Although TKN's predictor trains slower than Grid-Keypoint and Struct-VRNN because its Transformer encoder takes 750 epochs to reach the optima while Convlstm and VRNN take only 20 and 50, TKN's overall training speed is up to 2 to 3 times faster than the baselines as shown in Fig. 10.

B. Ablation Experiments

Keypoint Detector. We test the impact of convolution kernel size and find that it presents a much larger impact on reconstruction accuracy than on inference speed of keypoint detector

TABLE XI
COMPARISON BETWEEN USING EXPLICIT AND LATENT REPRESENTATION OF KEYPOINTS.

Method	SSIM	PSNR	Speed (ms)	FLOPs(G)	Params(M)
explicit	0.852	26.50	**6.7**	**0.01**	**0.7**
latent	**0.871**	**27.71**	8.3	0.19	18.9

as summarized in Table VII. Therefore we choose to use the 3×3 convolution kernel.

We then compare the influence of different structures and numbers of keypoints on reconstruction and prediction. Table VIII shows that TKN reconstructs frames better than the two Keypoints-based baselines and the sequential structure in Fig. 2(b), and achieves better performance with more keypoints. Meanwhile, the prediction module hits a performance bottleneck at 20 keypoints, indicating that too many keypoints pose difficulties for prediction. Fig. 11 shows that the keypoint detector of TKN performs better at capturing dynamic information than that of Struct-VRNN and Grid-keypoint on different actions.

Predictor relies on the results of the keypoint detector. We verified its performance by replacing its Transformer encoder with alternative modules, i.e., RNN [11], LSTM [16], GRU [7], and MLP, which are widely used in prediction tasks. We adjusted the input dimension to $\mathbb{R}^{d_{model}}$ (512), the number of layers to 6, and the dimension of the hidden layers to 2048, in all modules, for fair comparisons. We use the encapsulated RNN, LSTM, and GRU modules from PyTorch and use the MLP structure in Mlp-mixer [36]. As shown in Table IX, our predictor module presents comparable training and testing speeds with significantly higher accuracy and memory efficiency. It also shows the evaluated performance of the effects of different attention structures shown in Fig. 4. As we analyzed, the acceleration module we designed in Fig. 4(c) reduces the time and memory required for training and inference while ensuring accuracy.

Table X compares the performance of TKN's predictor employing the complete transformer structure with when using only its encoder part. We can see that the encoder-only method works much better in terms of both prediction speed and accuracy. This is because the transformer initially proposed for NLP problems requires embedding each word's label ID into a vector. Its translation process is similar to RNN's cycle principle which compares each high-dimensional output with the embedding vectors to get the word ID and then inputs the corresponding embedding vector to the transformer to translate the next word. In short, their input and output are finite discrete quantities, while the keypoints in our prediction task are continuous quantities that cannot be labeled with finite IDs, hence excluding the possibility of mapping the high-dimensional output to IDs. Moreover, each output in our task, which is the next input, is a floating point that cannot be acquired with 100% accuracy. Thus, the small errors in each transformer cycle are accumulated.

TKN employing the complete transformer has a long prediction time because the translation part of the transformer is a step-by-step process and each step goes through a complete transformer, while encoder-only TKN outputs all the results in one step.

Further, we test the impact of using explicit or latent representation of keypoints. As shown in Table XI, latent representation in high-dimension presents higher prediction accuracy. On the other hand, explicit representation only has limited speed improvement albeit it has fewer FLOPs and parameters.

In summary, the results validate that TKN greatly simplifies the complexity of prediction and accelerates the training and inference speed while improving the prediction accuracy.

REFERENCES

[1] A. K. Akan, E. Erdem, A. Erdem, and F. Güney, "Slamp: Stochastic latent appearance and motion prediction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 14 728–14 737.

[2] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, "Vivit: A video vision transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 6836–6846.

[3] A. Blattmann, T. Milbich, M. Dorkenwald, and B. Ommer, "Understanding object dynamics for interactive image-to-video synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 5171–5181.

[4] L. Castrejon, N. Ballas, and A. Courville, "Improved conditional vrnns for video prediction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7608–7617.

[5] C. Chen and P. Zhang, "Integrating cross-modal interactions via latent representation shift for multi-modal humor detection," in *Proceedings of the 3rd International on Multimodal Sentiment Analysis Workshop and Challenge*, 2022, pp. 23–28.

[6] X. Chen, W. Wang, J. Wang, and W. Li, "Learning object-centric transformation for video prediction," in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 1503–1512.

[7] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[8] E. L. Denton *et al.*, "Unsupervised learning of disentangled representations from video," *Advances in neural information processing systems*, vol. 30, 2017.

[9] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *2009 IEEE conference on computer vision and pattern recognition*. IEEE, 2009, pp. 304–311.

[10] 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," *arXiv preprint arXiv:2010.11929*, 2020.

[11] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[12] Z. Feng, J. Xu, L. Ma, and S. Zhang, "Efficient video transformers via spatial-temporal token merging for action recognition," *ACM Transactions on Multimedia Computing, Communications and Applications*, 2023.

[13] X. Gao, Y. Jin, Q. Dou, C.-W. Fu, and P.-A. Heng, "Accurate grid keypoint learning for efficient video prediction," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 5908–5915.

[14] S. Ge, F. Lin, C. Li, D. Zhang, W. Wang, and D. Zeng, "Deepfake video detection via predictive representation learning," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 18, no. 2s, pp. 1–21, 2022.

[15] V. L. Guen and N. Thome, "Disentangling physical dynamics from unknown factors for unsupervised video prediction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 474–11 484.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] H. Huang, Y. Huang, S. Xie, L. Lin, T. Ruofeng, Y.-w. Chen, Y. Li, and Y. Zheng, "Semi-supervised convolutional vision transformer with bi-level uncertainty estimation for medical image segmentation," in *Proceedings of the 31st ACM International Conference on Multimedia*, 2023, pp. 5214–5222.

[18] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, "Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1325–1339, jul 2014.

[19] T. Jakab, A. Gupta, H. Bilen, and A. Vedaldi, "Conditional image generation for learning the structure of visual objects," *methods*, vol. 43, p. 44, 2018.

[20] M. Khan, M. Saeed, A. El Saddik, and W. Gueaieb, "Artrivit: Automatic face recognition system using vit-based siamese neural networks with a triplet loss," in *2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE)*. IEEE, 2023, pp. 1–6.

[21] J. Li, W. Wang, J. Chen, L. Niu, J. Si, C. Qian, and L. Zhang, "Video semantic segmentation via sparse temporal transformer," in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, pp. 59–68.

[22] J. Liang, J. Cao, G. Sun, K. Zhang, L. Van Gool, and R. Timofte, "Swinir: Image restoration using swin transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 1833–1844.

[23] 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," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 012–10 022.

[24] Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin, and H. Hu, "Video swin transformer," *arXiv preprint arXiv:2106.13230*, 2021.

[25] ——, "Video swin transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 3202–3211.

[26] X. Man, D. Ouyang, X. Li, J. Song, and J. Shao, "Scenario-aware recurrent transformer for goal-directed video captioning," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 18, no. 4, pp. 1–17, 2022.

[27] M. Meng and J. Yu, "Zero-shot learning via robust latent representation and manifold regularization," *IEEE Transactions on Image Processing*, vol. 28, no. 4, pp. 1824–1836, 2018.

[28] M. Minderer, C. Sun, R. Villegas, F. Cole, K. P. Murphy, and H. Lee, "Unsupervised learning of object structure and dynamics from videos," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[29] M. Oliu, J. Selva, and S. Escalera, "Folded recurrent neural networks for future video prediction," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 716–731.

[30] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko, "Semi-supervised learning with ladder networks," *Advances in neural information processing systems*, vol. 28, 2015.

[31] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[32] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: a local svm approach," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 3. IEEE, 2004, pp. 32–36.

[33] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *Advances in neural information processing systems*, vol. 28, 2015.

[34] N. Srivastava, E. Mansimov, and R. Salakhudinov, "Unsupervised learning of video representations using lstms," in *International conference on machine learning*. PMLR, 2015, pp. 843–852.

[35] H. Tao, C. Hou, Y. Qian, J. Zhu, and D. Yi, "Latent complete row space recovery for multi-view subspace clustering," *IEEE Transactions on Image Processing*, vol. 29, pp. 8083–8096, 2020.

[36] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit *et al.*, "Mlp-mixer: An all-mlp architecture for vision," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24 261–24 272, 2021.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[38] T. Wang, H. Cheng, K. P. Chow, and L. Nie, "Deep convolutional pooling transformer for deepfake detection," *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 19, no. 6, pp. 1–20, 2023.

[39] Y. Wang, Z. Gao, M. Long, J. Wang, and S. Y. Philip, "Predrnn++: Towards a resolution of the deep-in-time dilemma in spatiotemporal predictive learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5123–5132.

[40] Y. Wang, L. Jiang, M.-H. Yang, L.-J. Li, M. Long, and L. Fei-Fei, "Eidetic 3d lstm: A model for video prediction and beyond," in *International conference on learning representations*, 2018.

[41] Y. Wang, M. Long, J. Wang, Z. Gao, and P. S. Yu, "Predrnn: Recurrent neural networks for predictive learning using spatiotemporal lstms," *Advances in neural information processing systems*, vol. 30, 2017.

[42] Y. Wang, H. Wu, J. Zhang, Z. Gao, J. Wang, S. Y. Philip, and M. Long, "Predrnn: A recurrent neural network for spatiotemporal predictive learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 2, pp. 2208–2225, 2022.

[43] Y. Wang, H. Wu, J. Zhang, Z. Gao, J. Wang, P. S. Yu, and M. Long, "Predrnn: A recurrent neural network for spatiotemporal predictive learning," *arXiv preprint arXiv:2103.09504*, 2021.

[44] X. Xiang, K. Zhang, Y. Qiao, and A. El Saddik, "Emhiformer: An enhanced multi-hypothesis interaction transformer for 3d human pose estimation in video," *Journal of Visual Communication and Image Representation*, vol. 95, p. 103890, 2023.

[45] G. Xu, J. Hao, L. Shen, H. Hu, Y. Luo, H. Lin, and J. Shen, "Lgvit: Dynamic early exiting for accelerating vision transformer," in *Proceedings of the 31st ACM International Conference on Multimedia*, 2023, pp. 9103–9114.

[46] Y. Xu, Z. Sun, Q. Li, Y. Sun, and S. Luo, "Active patterns perceived for stochastic video prediction," in *Proceedings of the 30th ACM International Conference on Multimedia*, 2022, pp. 5961–5969.

[47] Z. Xu, Z. Liu, C. Sun, K. Murphy, W. T. Freeman, J. B. Tenenbaum, and J. Wu, "Unsupervised discovery of parts, structure, and dynamics," *arXiv preprint arXiv:1903.05136*, 2019.

[48] S. Yang, L. Li, S. Wang, D. Meng, Q. Huang, and Q. Tian, "Structured stochastic recurrent network for linguistic video prediction," in *Proceedings of the 27th ACM International Conference on Multimedia*, 2019, pp. 21–29.

[49] G. Ying, Y. Zou, L. Wan, Y. Hu, and J. Feng, "Better guider predicts future better: Difference guided generative adversarial networks," in *Asian Conference on Computer Vision*. Springer, 2018, pp. 277–292.

[50] Y. Zhang, P. Tiwari, L. Rong, R. Chen, N. A. AlNajem, and M. S. Hossain, "Affective interaction: Attentive representation learning for multi-modal sentiment classification," *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 18, no. 3s, pp. 1–23, 2022.

[51] T. Zhou, S. Canu, P. Vera, and S. Ruan, "Latent correlation representation learning for brain tumor segmentation with missing mri modalities," *IEEE Transactions on Image Processing*, vol. 30, pp. 4263–4274, 2021.