Real-time Online Video Detection with Temporal Smoothing Transformers

Yue Zhao[1] and Philipp Krähenbühl[1]

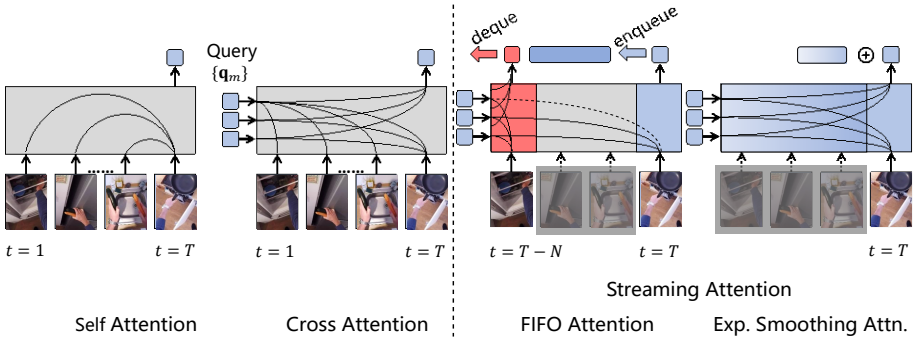
University of Texas at Austin, Austin TX 78712, USA
{yzhao,philkr}@cs.utexas.edu

Abstract. Streaming video recognition reasons about objects and their actions in every frame of a video. A good streaming recognition model captures both long-term dynamics and short-term changes of video. Unfortunately, in most existing methods, the computational complexity grows linearly or quadratically with the length of the considered dynamics. This issue is particularly pronounced in transformer-based architectures. To address this issue, we reformulate the cross-attention in a video transformer through the lens of kernel and apply two kinds of temporal smoothing kernel: A box kernel or a Laplace kernel. The resulting streaming attention reuses much of the computation from frame to frame, and only requires a constant time update each frame. Based on this idea, we build TeSTra, a Temporal Smoothing Transformer, that takes in arbitrarily long inputs with constant caching and computing overhead. Specifically, it runs 6× faster than equivalent sliding-window based transformers with 2,048 frames in a streaming setting. Furthermore, thanks to the increased temporal span, TeSTra achieves state-of-the-art results on THUMOS'14 and EPIC-Kitchen-100, two standard online action detection and action anticipation datasets. A real-time version of TeSTra outperforms all but one prior approaches on the THUMOS'14 dataset.

Keywords: Online action detection, action anticipation, transformer, temporal smoothing kernel

1 Introduction

The problem of online action detection [10] and anticipation [29] aims to determine what action is happening or will happen shortly at each time step without seeing the future. The challenge for online action detection is (1) how to effectively retain both the long-term trends and short-term cues when encoding the history and (2) how to eficiently compute at each time step in the stream-ing setting when the history gets longer. Recurrent models such as LSTM [22] and GRU [7] excel at updating the output recurrently but do not benefit from increasing sequence length due to the training dificulty [50]. Attention-based models [44], like Long Short-Term Transformer (LSTR) [54], are capable of handling sequences up to 8 minutes long with impressive prediction results. However, in the streaming setting, the attention computation of the long-term memory



	Self Attention	Cross Attention	FIFO Attention	Exp. Smoothing Attn.
Runtime	$O(T^2)$	$O(T)$	$O(1)$	$O(1)$
Memory	$O(T^2)$	$O(T)$	$O(N)$	$O(1)$

Fig. 1: A comparison of traditional attention computation (left) in streaming videos and our streaming attention (right). Unlike traditional approaches, our approach has a constant runtime per frame. Exponential smoothing attention has a constant memory footprint as well

has to be recomputed for each streaming window considered. Therefore, the computational cost per frame is proportional to the sequence length.

In this paper, we propose an effective and eficient approach, Temporal Smoothing Transformers (TeSTra), to encode suficiently long history with constant inference cost at each time step. TeSTra relies on an eficient attention that reuses much of the attention computation between consecutive frames. We reformulate attention through a kernel perspective [38,43] and explore two temporal kernels: a Box kernel and a Laplace kernel. Both kernels lead to an eficient streaming attention computation. A box kernel results in a First In First Out (FIFO) attention computation with a constant runtime update, but linear memory costs. A Laplace kernel results in an exponential smoothing attention with constant runtime and memory costs. Fig. 1 shows a comparison of traditional attention for streaming videos and our streaming attention. Both formulations exploit the fact that in streaming recognition queries used in cross attention are learned parameters and fixed during inference. During training, we use windowed attention in its original matrix multiplication form (with explicitly computed kernels). This allows us to enjoy all the GPU parallelism of modern transformer training. At test time, we switch to eficient streaming implementations.

To show the effectiveness of TeSTra, we conduct extensive experiments on standard benchmarks for online action detection and anticipation, namely THUMOS'14 [24] and EPIC-Kitchen-100 [9]. TeSTra achieves state-of-the-art performance on both benchmarks. Running at 142.8 FPS alone, TeSTra can serve as a building block for streaming video recognition with low latency. When we include an accelerated optical flow computing method and an image-based feature extractor, the overall system can run as fast as 41.1 FPS and achieves 67.3% mAP

on THUMOS'14, outperforming all but one prior approaches. Code is publicly available at https://github.com/zhaoyue-zephyrus/TeSTra/.

2 Related Work

Online Action Detection and Anticipation. Online action detection [10], also known as early action detection [21], aims to detect the start of an action in a video stream as soon as it happens. Much of prior work builds ever longer-term temporal reasoning using various recurrent units or networks [11,13,53]. Xu et al. [53] perform online detection (classification) on current frame and prediction the near-future actions simultaneously. StartNet [18] decomposes the online detection into two stages: action classification and start localization. The recently proposed LSTR [54] enlarges the effective temporal context to as long as 512 seconds by adopting the highly flexible cross-attention mechanism in Transformer [44]. However, the induced computation cost is proportional to the temporal span. In contrast, our streaming attention incurs the same constant runtime cost independent of temporal span.

Action anticipation [19], or forecasting [29], aims to predict the action before it occurs. Vondrick et al. [45] propose to anticipate by regressing the representations of future frames from past ones. Zeng et al. [58] and Rhinehart et al. [37] use inverse reinforcement learning to perform forecasting at multiple levels. For egocentric videos anticipation may additionally incorporate the camera wearer's trajectory [35], eye gaze [32], hand-object interaction [30], and environment affordance [34]. In this paper, we handle the problem by taking longer history into account, which is a general approach to both third-person and egocentric videos.

Transformers and its Eficient Variants. Since the Transformer architecture was introduced in [44], much work has gone into improving the eficiency of dot-product attention. Low-rank approximation on attention matrix [6,48] factorizes the attention matrix into two lower-rank matrices. Different eficient learnable sparsity patterns, such as locality-sensitive hashing [28], differentiable sorting [41] or fixed patterns [5,57], reduce the total number of attention operations. Query-based cross attention mechanisms compress longer-term input into a fixed-size representation via memory [36,31] or recurrence [8]. Based on a kernel-reformation [43], Katharopoulos et al. [27] propose linear attention by decomposing the kernel function $\kappa(q_m, k_n)$ between a query-key pair into a product between the feature mapping of query and key, i.e. $\varphi(q_m)^\top \cdot \varphi(k_n)$. In computer vision, Transformers are made more eficient by (1) leveraging hierarchy using shifted local window [33] and pooling attention [14], (2) applying axial attention on separate dimensions [46], and (3) using asymmetric attention (cross attention) to squeeze high-dimensional inputs into tighter latent variables [26]. In speech recognition, transformers are tailored to streaming decoding by integrating recurrence [61] or memory [52]. In this paper, we follow the kernel interpretation of Tsai et al. [43], and show how to eficiently update streaming attention kernels.

Eficient Video Processing. Videos are notoriously expensive to process. TSN [47] suggests sampling frames sparsely and running 2D CNNs on the se-

lected frames. MVCNN [59] and CoViAR [51] directly learn video representation from compressed videos. X3D [15] and CSN [42] reduce computation FLOPs by leveraging channel-wise separable convolution. However, 3D CNN takes video clips as input whose span can be $2-3$ seconds, therefore may not be the best solution in a low-latency application. Par-Inception [3] tackles the latency issue by introducing depth-parallelism to the vanilla I3D [4] at increased implementation dificulty. Most of the previous methods focus on trimmed videos whose duration is often in several seconds while our method focuses on streaming videos whose length can be as long as hours. However, many of these 3D CNNs may form a good backbone to our system.

3 Preliminaries

Attention. The attention mechanism [44] is a weighted addition of the input features. The weights are guided by the similarities between the key and query elements on an input sequence:

$$\text{Attention}(Q, X) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{C}}\right) \cdot V = \text{Softmax}\left(\frac{Q \cdot (XW_k)^\top}{\sqrt{C}}\right) \cdot XW_v, \tag{1}$$

where $Q \in R^{M \times C}$ is a set of M queries, $X = \cdots x_n \cdots^\top \in R^{N \times d}$ is the sequence of N input tokens, $W_{k/v} \in R^{d \times C}$ is the weight to map the input to key/value vector and C is the feature dimension of $x^\top W_k$. For self-attention computes queries from the inputs sequence $Q = XW_q^n$ ($M = N$ in this case). Cross-attention uses a queries Q that do not relate to the input sequence X (generally $M = N$ in this case). Cross-attention is commonly used in the encoder-decoder architecture [44]. Cross-attention with $M \ll N$ is also used to eficiently encode large amounts of data into a fixed-size representation [26,54].

Attention as kernels. The distance computation in attention is similar to the mechanism of kernel learning [38]. Tsai et al. [43] reformulated Eq. (1) from the perspective of kernels:

$$\text{Attention}(q_m, \{x_n\}) = \frac{\sum_{n=1}^{N} \kappa(q_m, k_n) v_n}{\sum_{n=1}^{N} \kappa(q_m, k_n)}, \tag{2}$$

where $\kappa(\cdot, \cdot) : R^C \times R^C \to R^+$ is a generalized kernel function, which depicts the similarity between the pair of input vectors. Eq. (1) is equivalent to Eq. (2) for a kernel $\kappa(q_m, k_n) = \exp(\frac{q_m^\top k_n}{\sqrt{C}})$. In the next section, we show that this kernel perspective leads to an eficient streaming formulation of attention in the context of streaming video recognition.

4 Eficient Attention on streaming input

We use cross-attention to summarize a large stream of past frames into a fixed size context representation. We use a fixed number learned queries and variable number of keys and values from past frames as input. See Fig. 1 for an example.

In streaming tasks, we are constantly receiving input and want to generate the corresponding output on the fly. Examples include simultaneous interpretation or online detection in broadcast videos. Let $x_{[1:t]} = \{x_1, x_2, \cdots, x_t\}$ denote a sequence of encoded past video frames for the current time-step t. The encoder may use an image-based [20,25] or short-clip-based [4,15] CNN. Top-performing video models [54] summarize large parts of the video through cross-attention on either the entire sequence, i.e. Attention($q_1 \dots q_M, x_{[1:t]}$) or a chunk of input by sliding a N-sized temporal window, i.e. Attention($q_1 \dots q_M, x_{[t-N+1:t]}$). Mathematically, this attention operation is captured in Eq. (2). Here, a small number of queries $\{q_1 \dots q_M\}$ summarize a large temporal context. Queries combine learned parameters $\{\lambda_1 \dots \lambda_M\}$ with a temporal embedding ω_t of the current frame: $q_m = \lambda_m + \omega_t$. Keys $\{k_1 \dots k_t\}$ combine a frame-level embeddings $f_n = W_k x_n$ with a temporal embedding ω_t: $k_n = f_n + \omega_n$. Values $\{v_1 \dots v_t\}$ use the same frame-level features $v_n = W_v x_n$. In this setup, keys and values of past frames remain unchanged, learned queries are constant during inference, only the temporal query embedding changes frame to frame. This changing tem-poral embedding does change the attention kernel κ for each new frame. This means in a streaming setting, we have no choice but to recompute the entire at-tention operation frame after frame. This recomputation grows linearly with the size N of the temporal context considered. Next, we show how a reformulation of the attention mechanism leads to a much more eficient streaming evaluation.

Streaming Attention. Note, that both queries and keys combine a temporal and feature-level embedding in their distance kernel $\kappa(q_m, k_n) = \kappa(\lambda_m + \omega_t, f_n + \omega_n)$. In Streaming Attention, we simple split this kernel into temporal and feature component: $K(\omega_t, \omega_n)\kappa(\lambda_m, f_n)$. The Streaming Attention operation reduces to

$$\text{Stream-Attention}(q_m, x_{[1:t]}) = \frac{\sum_{n=1}^{t} K(\omega_t, \omega_n)\kappa(\lambda_m, f_n)v_n}{\sum_{n=1}^{t} K(\omega_t, \omega_n)\kappa(\lambda_m, f_n)}. \tag{3}$$

Most of the features and kernels used in this attention block remain constant throughout the streaming setting. Moving from timestep t to $t+1$ only changes the temporal kernel $K(\omega_t, \omega_n)$ to $K(\omega_{t+1}, \omega_n)$ and adds one more element (f_{t+1}, v_{t+1}). Because of the change in the temporal kernel, a naive evaluation of streaming attention (3) still requires a linear runtime in the size of the temporal context. However, the right choice of a temporal kernel can alleviate this. Here, we explore two kernels: A box (or uniform) kernel $K_B(\omega_t, \omega_n) = 1_{[t-n<N]}$ and a Laplace kernel $K_L(\omega_t, \omega_n) = \exp(-\lambda(t-n))$ for $\lambda > 0$. Each of these kernels leads to an eficient streaming attention mechanism. A box kernel results in first-in-first-out (FIFO) attentionw while a Laplace kernel leads to exponential smoothing attention. Fig. 2 provides an overview of both kernels.

FIFO Attention. Let us define the numerator and denominator of Eq. (3) to be two intermediate variables

$$\text{Stream-Attention}(q_m, x_{[1:t]}) = \frac{\varphi(t)}{\psi(t)}. \tag{4}$$

(a) Box kernel (b) Laplace kernel

Fig. 2: A visualization of a box kernel (a) and Laplace kernel (b) and their streaming computation

Both $\varphi(t) = \sum_{n=1}^{t} K_B(\omega_t, \omega_n)\kappa(\lambda_m, f_n)v_n$ and $\psi(t) = \sum_{n=1}^{t} K_B(\omega_t, \omega_n)\kappa(\lambda_m, f_n)$ are updated by the following recursion as the streaming attention progresses:

$$\begin{aligned} \varphi(t+1) &= \varphi(t) + \kappa(\lambda_m, f_t)v_t - \kappa(\lambda_m, f_{t-N})v_{t-N} \\ \psi(t+1) &= \psi(t) + \kappa(\lambda_m, f_t)v_t - \kappa(\lambda_m, f_{t-N}), \end{aligned} \tag{5}$$

where $\kappa(\lambda_m, f_{t-N}) = 0$ and $v_{t-N} = 0$ for $t \leq N$, $\varphi(0) = 0$ and $\psi(0) = 0$.

Like a FIFO queue, we keep track of $\varphi(t)$ and $\psi(t)$ and update them by subtracting the quantity contributed by the input at time $(t - N)$ and adding up the one at time t in the long run. Therefore, we call this formulation FIFO-Attention. The advantage of FIFO-Attention is that the computational cost becomes $O(MC)$ for M queries and values of C channels. Neither the effective window size N nor the actual time-step t influences the runtime. However, the subtraction operation in Eq. (5) requires us to keep a window of features and kernel values in memory. Hence, the memory complexity is still $O(N)$. The Laplace kernel addresses this issue.

Exponential Smoothing Attention. The Laplace kernel K_L allows for an even more eficient recursive update:

$$\begin{aligned} \tilde{\varphi}(t) &= e^{-\lambda}\tilde{\varphi}(t-1) + \kappa(\lambda_m, f_t)v_t \\ \tilde{\psi}(t) &= e^{-\lambda}\tilde{\psi}(t-1) + \kappa(\lambda_m, f_t), \end{aligned} \tag{6}$$

where $\tilde{\varphi}(0) = 0$ and $\tilde{\psi}(0) = 0$. The parameters λ controls the temporal ex-tent of the attention. The above operation (6) is known as exponential smooth-ing [23]. Therefore we name this attention Exponential Smoothing Attention, or ES-Attention for short. Both ES- and FIFO-Attention reduce to the same opera-tion if $\lambda = 0$ and the windows size $N \to \infty$. The time complexity of ES-Attention is also constant in the temporal window considered $O(MC)$. More importantly, the space complexity reduces from $O(N)$ to $O(1)$ since we only maintain ψ, $\tilde{\varphi}$ and no longer keep values in our window around. Exponential smoothing instead slowly reduces the influence of older keys and values in the attention.

Video recognition with streaming attention. The streaming attention can replace the vanilla cross-attention in current Transformer architectures with min-

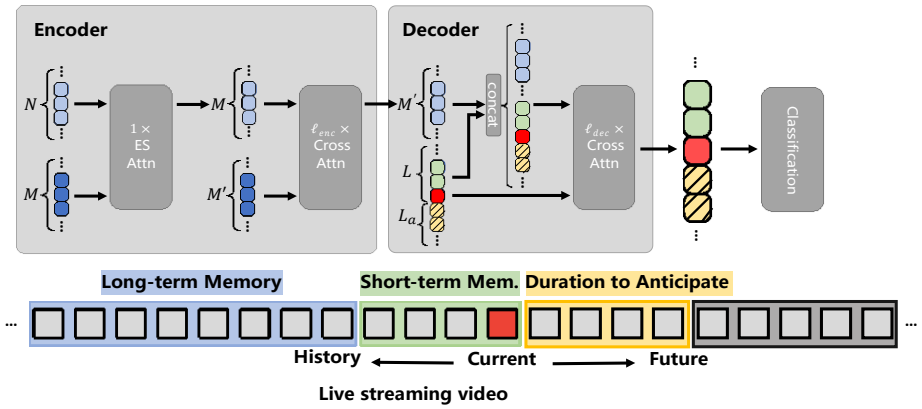


Fig. 3: Overview of our streaming attention architecture TeSTra. The basic setup follows LSTR [54]: A long-term memory compresses a long temporal history into M representative queries. A short-term attention mechanism uses the compressed memory and a short history of frames to compute current and future actions. The main advantage of TeSTra is that the long-memory incurs only constant cost, and thus allows for much more eficient long-term reasoning

imal modification. Specifically, we follow LSTR architecture [54] for all our experiments, due to its state-of-the-art performance on online action detection. The overall architecture of TeSTra is sketched in Fig. 3. Given a sequence of encoded vectors $x_{[1:t]} = \{x_1, x_2, \cdots, x_t\}$, where t refers to the current time stamp, we divide the historic frames into two parts: short-term memory $x_{[t-L+1:t]}$ if size $L \leq 32$ and long-term memory which contains the rest of distant inputs, namely $x_{[1:t-L]}$. The architecture follows an encoder-decoder [44,54] design. The encoder module encodes the long-term memory into $M = 16$ query features. The decoder uses the query features and short-term memory to predict current and anticipated actions.

The encoder has two stages of memory compression. First, it uses an ES-Attention-based Transformer decoder unit [44] to compress the long-term memory into M latent vectors Z using learnable queries Q.

$$\begin{aligned} Q' &= \text{Attention}(Q, Q), \\ Z' &= \text{ES-Attention}(\sigma(Q'), X_{[1:t-L]}), \\ Z &= \text{FFN}(\sigma(Z')), \end{aligned} \tag{7}$$

where σ denotes the nonlinear mapping which is composed of a skip connection with Q followed by a LayerNorm [1]. Next, the compressed vectors are further cross-attended by M' learnable queries through ℓ_{enc} decoder units into $Z_{\ell_{\text{enc}}}$. Strictly speaking, it should be possible to learn Q' directly. However, the training dynamics of transformer work out better using a self-attention block first. Fig. 4 shows an overview of the encoder.

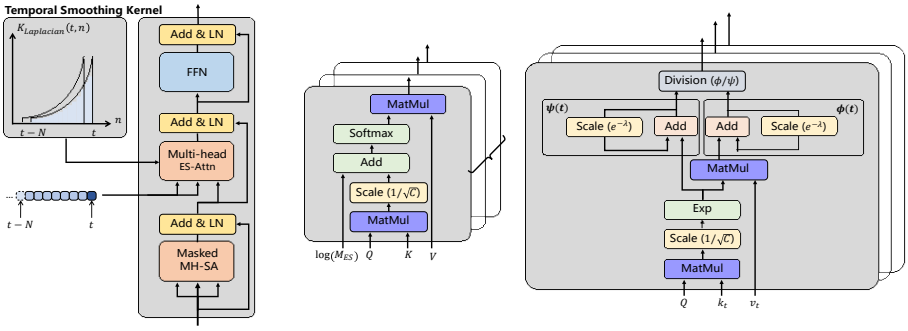


Fig. 4: The basic building blocks of TeSTra. Left: the Transformer Encoder with ES-Attention; Middle: Multi-head ES-Attention at training time; Right: Multi-head ES-Attention at inference time

The decoder uses the short-term memory as queries to attend the compressed memory and retrieve relevant information through a stack of ℓ_{dec} decoder units.

$$\begin{aligned} X'_{[t-L+1:t]} &= \text{Attention}(X_{[t-L+1:t]}, X_{[t-L+1:t]}), \\ O' &= \text{Attention}(X'_{[t-L+1:t]}, Z_{\ell_{enc}} \,\|\, X_{[t-L+1:t]}), \\ O &= \text{FFN}(\sigma(O')), \end{aligned} \tag{8}$$

In Eq. (8), we construct the key/value tokens by concatenating $[\cdot \,\|\, \cdot]$ both the compressed long-term memory and short-term memory to incorporate all the known historic information. This proves to be effective for action anticipation, where the closer memory is more important to indicate the upcoming action. The L output vectors are then passed through a linear layer to produces the scores $s_{[t-L+1:t]} \in \mathbb{R}^{L\times(K+1)}$ over K action classes plus one non-action (background) class [1]. At inference time, we take the score s_t to be online detection result. In action anticipation, the frames in the anticipating duration are not observable. We thus attach L_a learnable tokens after short-term memory predict L_a anticipated actions $s_{[t+1:t+L_a]}$.

Training TeSTra. At inference time, we naturally apply the recursion in Eq. (6) in the streaming setting. During training, however, it is computationally inefficient to feed all historic inputs and update them recursively on a modern GPU architecture. To handle this, we cut the video into a clip $x_{t-L-N+1:t}$. Multiple clips share the same length N and thus can be packed into a batch. Furthermore,

[1] $s_t \in \mathbb{R}^K$ if the background class is absent.

instead of recursion, we compute the attention in matrix form:

$$\text{ES-Attention}_{\text{train}}(Q_{,,}, X) = \text{Softmax}\left(\log(M_{ES}) + \frac{QK^{,,}}{C}\right) \cdot V, \tag{9}$$

$$M_{E_s} = \begin{bmatrix} e^{-\lambda(N-1)} & e^{-\lambda(N-2)} & \dots & 1 \\ e^{-\lambda(N-1)} & e^{-\lambda(N-2)} & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ e^{-\lambda(N-2)} & \dots & & 1 \end{bmatrix}, \tag{10}$$

where $\log(\cdot)$ takes the element-wise logarithm of a matrix and the exponential smoothing matrix $M_{ES} \in R^{M \times N}$ is a Vandermonde matrix. Since we train on the windowed input and test on un-windowed streaming input, we select a decay factor λ and window size N such that $e^{-\lambda(N-1)}$ is suficiently small. This minimizes the effect of a potential train-test gap. Fig. 4 shows the difference between training and inference for streaming attention.

We use the cross-entropy loss to predict both current and anticipated actions. Following [54,19], we predict actions for all frames in short-term memory for a stronger supervisory signal. We use a causal attention mask [44] on the short-term memory to avoid future actions from influencing our predictions.

5 Experiments

5.1 Experimental Setup

Datasets. We conduct experiments on THUMOS'14 [24] and Epic-Kitchen-100 (EK100) [9]. THUMOS'14 contains 413 untrimmed videos annotated with 20 actions. We train our model on the validation set (200 videos) and evaluate on the test set (213 videos). Epic-Kitchen-100 contains 100 hours of egocentric videos with 90K action segments. The narrations are mapped into 97 verb classes and 300 noun classes. We follow the train/val split given by Furnari et al. [16].
Evaluation Metrics. For THUMOS'14, we measure the performance of both online action detection and anticipation with per-frame mean average precision (mAP). Anticipation mAP uses an anticipation period τ_o which varies from 0.25s to 2.0s with a stride of 0.25s. Online detection mAP is as a special case of anticipation mAP at $\tau_o = 0$. EK-100 uses mean Top-5 Verb/Noun/Action Recall to measure anticipation performance per instance with a predefined $\tau_o = 1$s [9].
Implementation Details. On THUMOS14, we pre-process the videos into 24 FPS, extract the two-stream deep features pretrained on ActivityNet or Kinetics following Xu et al. [54]. The visual stream is a ResNet-50 [20] while the motion stream uses BN-Inception [25]. On EK100, we first pre-process the videos into 30 FPS and fine-tune the two-stream TSN [47] on EK100 action classification task with ImageNet-pretrained parameters, following Furnari et al. [16]. When training TeSTra, we apply equalization loss [40] to handle the long-tailness of actions. Our model is not restricted to using 2D CNNs as backbone. Eficient 3D

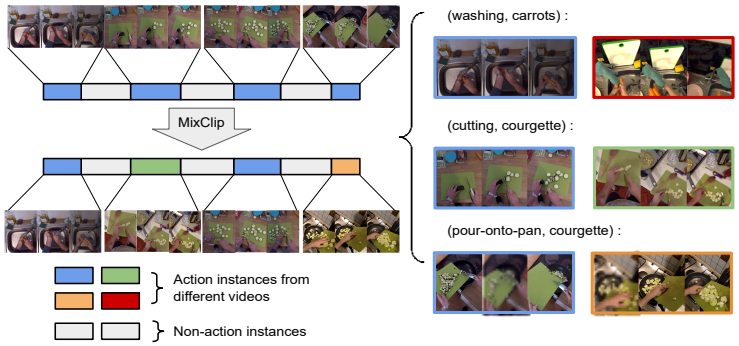


Fig. 5: Illustration of MixClip. In the example sequence, we have 4 action instances and 2 of them are replaced by another clip that comes from another video but is annotated with the same action category

CNN such as X3D [15] is also applicable but the longer input span might cause higher latency.

The training procedure of TeSTra on THUMOS'14 follows Xu et al. [54] for fair comparison. Specifically, we train TeSTra with batch size of 16 for 25 epochs using Adam optimizer with a weight decay of 5e-5 and a base learning rate of 7e-5. We apply a cosine annealing schedule with linear warm-up, i.e. the learning rate linearly increases from 0 to 7e-5 in the first 10 epochs and then decays following a cosine function.

MixClip. The model takes as input both short clips as working memory and frames in the longer history as long-term memory. The duration of long history is significantly larger (often 10×) than the short clip. This means that two neighboring clips of interest share a large portion of historical frames. This causes the model to overfit to those scene-related cues and fail to generalize to unseen scenarios. To resolve this, we propose a simple augmentation technique called MixClip which increases the diversity of long history by composing short clips from different recordings into each other.

Assume that the long memory is composed of a sequence of action instances $\{(t_i^{(s)}, t_i^{(e)}, a_i)\}$, where $t_i^{(s)}, t_i^{(e)}$ denotes the start and end time while a_i denotes the action label. With probability p_{mc}, each of the action instances may be replaced with another instance with the same label from a different video. This input feature sequence is randomly cropped if the new instance's duration longer. Otherwise, the input feature sequence is padded to ensure that the length of history is unchanged for ease of implementation. Fig. 5 gives an illustration.

MixClip is inspired by some popular augmentation techniques widely used in image classifications, such as CutOut [12], Mixup [60], and CutMix [55].

5.2 Main Results

THUMOS'14. We conduct both online action detection and anticipation experiments on THUMOS'14. In both tasks, the backbone network from which the

Table 1: Result of online action detection on THUMOS'14. denotes optical flow computed by NVIDIA Optical Flow SDK, a faster alternative to TV-L1 [56]. More detailed runtime analysis will be provided in Sec. 5.4

(a) Using ANet-pretrained feature

Method	mAP
RED [17]	45.3
IDN [13]	50.0
TRN [53]	47.2
OadTR [49]	58.3
LSTR [54]	65.3
Ours	68.2

(b) Using Kinetics-pretrained feature

Method	mAP
IDN [13]	60.3
TRN [53]	62.1
OadTR [49]	65.2
LSTR [54]	69.5
Ours	67.3
Ours	71.2

Table 2: Result of online action anticipation on THUMOS'14. † was reproduced by us because LSTR [54] only reported ActivityNet-pretrained results

method	Pre-train	mAP@τ_o								average
		0.25	0.50	0.75	1.0	1.25	1.50	1.75	2.0	
RED [17]	ANet1.3	45.3	42.1	39.6	37.5	35.8	34.4	33.2	32.1	37.5
TRN [53]		45.1	42.4	40.7	39.1	37.7	36.4	35.3	34.3	38.9
TTM [49]		45.9	43.7	42.4	41.0	39.9	39.4	37.9	37.3	40.9
LSTR [54]		-	-	-	-	-	-	-	-	50.1
Ours		64.7	61.8	58.7	55.7	53.2	51.1	49.2	47.8	55.3
TTM [49]	K400	46.8	45.5	44.6	43.6	41.9	41.1	40.4	38.7	42.8
LSTR† [54]		60.4	58.6	56.0	53.3	50.9	48.9	47.1	45.7	52.6
Ours		66.2	63.5	60.5	57.4	54.8	52.6	50.5	48.9	56.8

feature is extracted is pretrained on either ActivityNet v1.3 [2] or Kinetics [4]. Table 1 shows the results of online action detection. TeSTra surpasses the previous states-of-the-art by a large margin. We also adopt NVIDIA Optical Flow SDK (NVOFA) [2] for faster optical flow computation. NVOFA can run as fast as 1K FPS on a 240×180 image sequence on a modern GPU. We denote the model that takes NVOFA optical flow as input to be TeSTra . We observe some performance drop, but an mAP of 67.3% is still competitive. Most importantly, the runtime of the entire pipeline is significantly sped up. More detailed discussion on runtime analysis will be provided in Sec. 5.4. All results use ES-Attention.

Table 2 shows the results of online action anticipation. TeSTra with Kinetics-pretrained feature achieves an average mAP of 56.8%, outperforming all previous methods. For fair comparison, we also rerun LSTR [54] using the same Kinetics-pretrained feature. This improved LSTR is still 4% below TeSTra.

[2] https://developer.nvidia.com/opticalflow-sdk

Table 3: Result of action anticipation on EK100. The upper half lists RGB-only methods; in lower half all types of inputs are allowed

Method	Input	Pre-train	overall			unseen			tail		
			verb	noun	action	verb	noun	action	verb	noun	action
RULSTM [16]	RGB	IN-1k	27.5	29.0	13.3	29.8	23.8	13.1	19.9	21.4	10.6
AVT [19]		IN-1k	27.2	30.7	13.6	-	-	-	-	-	--
AVT [19]		IN-21k	30.2	31.7	14.9	-	-	-	-	-	
Ours		IN-1k	26.8	36.2	17.0	27.1	30.1	13.3	19.3	28.6	13.7
RULSTM [16]	RGB +OF +Obj	IN-1k	27.8	30.8	14.0	28.8	27.2	14.2	19.8	22.0	11.1
TempAgg [39]		IN-1k	23.2	31.4	14.7	28.0	26.2	14.5	14.5	22.5	11.8
AVT+ [19]		IN-1k	25.5	31.8	14.8	25.5	23.6	11.5	18.5	25.8	12.6
AVT+ [19]		IN-21k	28.2	32.0	15.9	29.5	23.9	11.9	21.1	25.8	14.1
Ours	RGB+OF	IN-1k	30.8	35.8	17.6	29.6	26.0	12.8	23.2	29.2	14.2

EK100. We compare TeSTra with prior works on the EPIC-Kitchen-100 action anticipation track [9] in Table 3. We split the results into two halves: the upper half contains methods with only RGB inputs and the lower half uses additional information, such as optical flow and object feature. Using the same ImageNet-1k-pretrained feature, TeSTra significantly outperforms RULSTM [16] and AVT [19] on the action-level recall. The improvement is most pronounced in the increase noun-level recall. This demonstrates the effectiveness of incorporating longer input for anticipation. The long-memory recalls many objects that appeared previously. TeSTra with RGB+OF achieves 4.4% higher verb-level recall than TeSTra with only RGB. One reason for this our early-fusion. Unlike late-fusion approaches, RULSTM and AVT+, we concatenate RGB and optical-flow feature at the beginning so that motion-related feature can be more effectively leveraged. Again, all results use ES-Attention.

5.3 Ablation Studies

We conduct ablation experiments on EK100 to study the role of each module in the architecture. Our full ablations uses the RGB-only model, but conclusions generally hold for two-stream input as well.

Temporal Smoothing Kernels. We first verify the correctness of the temporal smoothing kernels at inference time in Table 4. If we apply the box kernel and apply the FIFO recursion defined in Eq. (5), the result is 16.14%. However, if we use the exponential smoothing recursion defined in Eq. (6) with decay factor $\lambda = 0$, action recall drops by 0.2 ? 0.4% on unseen and tail classes. This indicates the necessity to cache historic elements and pop them when the queue becomes full. When using the Laplace kernel, we compare batch mode where windowed attention is computed using Eq. (9) and stream mode where exponential smoothing recursion is computed using Eq. (6). The results are consistent (less than 0.05% difference).

Table 4: Temporal smoothing kernels. Using explicit windowed-attention and stream-attention under the Laplace kernel yield consistent results

Kernel Type	Test Mode	overall act. rec.	unseen act. rec.	tail act. rec.
Box	FIFO (Eq. (5))	16.14	12.64	12.89
Box	ES (Eq. (6); $\lambda = 0$)	16.08	12.22	12.70
Laplace	ES (Eq. (9))	16.95	13.33	13.73
Laplace	ES (Eq. (6))	16.94	13.28	13.72

Table 5: Ablation studies on position embeddings. Temporal position embeddings are unnecessary for long-term memory, justifying our design of separate the temporal smoothing kernel and feature vector

PE @ long memory	PE @ short memory	overall act. rec.
✗	✓	17.0
✓	✓	16.8
✗	✗	15.7

Effectiveness of Positional Embedding. The rationale behind streaming attention is that we can separate the attention kernel into temporal and feature components. To justify this, we add a temporal positional embedding in the long-term memory and observe no performance improvement from Table 5. We also try to remove the temporal embedding in the short-term memory but this changes the result significantly (-1.3%).

Effectiveness of MixClip Table 6a shows the effect of MixClip rate on the anticipation result. When no MixClip is applied, the baseline drops to 15.5% action recall. The performance consistently improves with MixClip and achieves the best (17.0%) at $p_{mc} = 0.5$.

Fusing long- and short-term memory Table 6b compares different ways of fusing long- and short-term memory. The naive way is to treat long- and short-term memory separately, i.e. (1) use the TeSTra encoder to compress distant inputs and (2) use closer inputs as queries in the TeSTra decoder to attend to this compressed set of vectors. We observe that this no-fuse approach achieves 15.9% which is even 0.2% lower than short-memory-only baseline, where $N = 0$ and the TeSTra decoder is instantiated by self-attention. This indicates that we might need to incorporate the relationship within the short-term memory too. To achieve this, we try to augment long-term memory by attaching short-term memory, denoted by "@ long mem.", but see no significant improvement. It might be because long-term memory is much longer than the short-term one so that the short-term information is overwhelmed at the first stage of compression. Since the memory length after compressed is in the same order as the short-term memory, we concatenate both ("@ comp. mem.") and get 17.0% action recall, improving the naive way by 1.1%.

Table 6: Ablation studies on MixClip and long-/short-term memory fusing

(a) The effect of MixClip

MixClip Rate	0	0.2	0.5	0.8
overall v. rec.	25.8	26.0	26.8	26.2
overall n. rec.	34.6	35.3	36.2	35.2
overall act. rec.	15.5	16.0	17.0	16.2

(b) Long- and short-term memory fusion

How to fuse	overall act. rec.
w/o. long mem.	16.1
no fuse	15.9
@ long mem.	16.0
@ comp. mem.	17.0

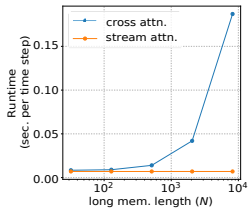


Fig. 6: Runtime comparison between vanilla cross attention and our exponential smoothing attention

	OF Comp.	RGB Feat.	OF Feat.	TeSTra	Total
Ours	1,000	150.0	104.7	142.8	41.1
Ours	19.3				12.6

Table 7: Runtime profile (in FPS) for the entire detection system. Real-time TeSTra uses NVOFA optical-flow while the default one uses TV-L1 [56]

5.4 Runtime Analysis

Finally, we study the runtime speed of TeSTra using an NVIDIA Quadro RTX 6000 GPU. Fig. 6 shows the comparison of inference speed between LSTR with cross attention and TeSTra with ES-Attention. We choose the length of the long memory N to be $\{32, 128, 512, 2048, 8196\}$. We can clearly see that the runtime per time step scales linearly for cross-attention-based LSTR but keeps constant for TeSTra. Specifically, TeSTra runs at a speed of 142.8 FPS. If we integrate TeSTra into the online detection system, we need to take into account of the computation overhead by the optical flow computation and feature extraction. The runtime profile is summarized in Table 7. The full TeSTra runs at 12.6 FPS. The bottleneck is computing optical flow using TV-L1 algorithm [56]. Using the NVOFA, the real-time TeSTra can run at 41.1 FPS.

6 Conclusion

We propose stream attention based on the kernel-based reformulation of cross-attention and apply two kinds of temporal smoothing kernels that reduce the inference computation to constant cost per frame. The resultant temporal smoothing transformer achieves excellent performance while running at a low latency. We hope that our design can shed some light on developing more eficient models for long-term videos understanding.

Acknowledgement This material is in part based upon work supported by the National Science Foundation under Grant No. IIS-1845485, IIS-2006820, and the NSF Institute for Foundations of Machine Learning.

References

1. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)
2. Caba Heilbron, F., Escorcia, V., Ghanem, B., Carlos Niebles, J.: Activitynet: A large-scale video benchmark for human activity understanding. In: CVPR (2015)
3. Carreira, J., Patraucean, V., Mazare, L., Zisserman, A., Osindero, S.: Massively parallel video networks. In: ECCV (2018)
4. Carreira, J., Zisserman, A.: Quo vadis, action recognition? a new model and the kinetics dataset. In: CVPR (2017)
5. Child, R., Gray, S., Radford, A., Sutskever, I.: Generating long sequences with sparse transformers. arXiv preprint arXiv:1904.10509 (2019)
6. Choromanski, K.M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J.Q., Mohiuddin, A., Kaiser, L., et al.: Rethinking attention with performers. In: ICLR (2021)
7. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. In: Deep Learning and Representation Learning Workshop (2014)
8. Dai, Z., Yang, Z., Yang, Y., Carbonell, J.G., Le, Q., Salakhutdinov, R.: Transformer-xl: Attentive language models beyond a fixed-length context. In: ACL (2019)
9. Damen, D., Doughty, H., Farinella, G.M., , Furnari, A., Ma, J., Kazakos, E., Moltisanti, D., Munro, J., Perrett, T., Price, W., Wray, M.: Rescaling egocen-tric vision: Collection, pipeline and challenges for epic-kitchens-100. IJCV (2021), https://doi.org/10.1007/s11263-021-01531-2
10. De Geest, R., Gavves, E., Ghodrati, A., Li, Z., Snoek, C., Tuytelaars, T.: Online action detection. In: ECCV (2016)
11. De Geest, R., Tuytelaars, T.: Modeling temporal structure with lstm for online action detection. In: WACV (2018)
12. DeVries, T., Taylor, G.W.: Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552 (2017)
13. Eun, H., Moon, J., Park, J., Jung, C., Kim, C.: Learning to discriminate information for online action detection. In: CVPR (2020)
14. Fan, H., Xiong, B., Mangalam, K., Li, Y., Yan, Z., Malik, J., Feichtenhofer, C.: Multiscale vision transformers. In: ICCV (2021)
15. Feichtenhofer, C.: X3D: Expanding architectures for eficient video recognition. In: CVPR (2020)
16. Furnari, A., Farinella, G.M.: Rolling-unrolling lstms for action anticipation from first-person video. TPAMI (2020)
17. Gao, J., Yang, Z., Nevatia, R.: Red: Reinforced encoder-decoder networks for action anticipation. In: BMVC (2017)
18. Gao, M., Xu, M., Davis, L.S., Socher, R., Xiong, C.: Startnet: Online detection of action start in untrimmed videos. In: ICCV (2019)
19. Girdhar, R., Grauman, K.: Anticipative video transformer. In: ICCV (2021)
20. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
21. Hoai, M., De la Torre, F.: Max-margin early event detectors. IJCV 107(2), 191–202 (2014)
22. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735–1780 (1997)

23. Holt, C.C.: Forecasting seasonals and trends by exponentially weighted moving averages. International journal of forecasting 20(1), 5–10 (2004)
24. Idrees, H., Zamir, A.R., Jiang, Y., Gorban, A., Laptev, I., Sukthankar, R., Shah, M.: The THUMOS challenge on action recognition for videos "in the wild". CVIU (2016), http://arxiv.org/abs/1604.06182
25. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML (2015)
26. Jaegle, A., Gimeno, F., Brock, A., Vinyals, O., Zisserman, A., Carreira, J.: Perceiver: General perception with iterative attention. In: ICML (2021)
27. Katharopoulos, A., Vyas, A., Pappas, N., Fleuret, F.: Transformers are rnns: Fast autoregressive transformers with linear attention. In: ICML (2020)
28. Kitaev, N., Kaiser, L., Levskaya, A.: Reformer: The eficient transformer. In: ICLR (2020)
29. Kitani, K.M., Ziebart, B.D., Bagnell, J.A., Hebert, M.: Activity forecasting. In: ECCV (2012)
30. Koppula, H., Saxena, A.: Learning spatio-temporal structure from rgb-d videos for human activity detection and anticipation. In: ICML (2013)
31. Lei, J., Wang, L., Shen, Y., Yu, D., Berg, T., Bansal, M.: Mart: Memory-augmented recurrent transformer for coherent video paragraph captioning. In: ACL (2020)
32. Li, Y., Liu, M., Rehg, J.M.: In the eye of beholder: Joint learning of gaze and actions in first person video. In: ECCV (2018)
33. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: ICCV (2021)
34. Nagarajan, T., Li, Y., Feichtenhofer, C., Grauman, K.: Ego-topo: Environment affordances from egocentric video. In: CVPR (2020)
35. Park, H.S., Hwang, J.J., Niu, Y., Shi, J.: Egocentric future localization. In: CVPR (2016)
36. Rae, J.W., Potapenko, A., Jayakumar, S.M., Hillier, C., Lillicrap, T.P.: Compressive transformers for long-range sequence modelling. In: ICLR (2020)
37. Rhinehart, N., Kitani, K.M.: First-person activity forecasting with online inverse reinforcement learning. In: ICCV (2017)
38. Schölkopf, B., Smola, A.J., Bach, F., et al.: Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press (2002)
39. Sener, F., Singhania, D., Yao, A.: Temporal aggregate representations for long-range video understanding. In: ECCV (2020)
40. Tan, J., Wang, C., Li, B., Li, Q., Ouyang, W., Yin, C., Yan, J.: Equalization loss for long-tailed object recognition. In: CVPR (2020)
41. Tay, Y., Bahri, D., Yang, L., Metzler, D., Juan, D.C.: Sparse sinkhorn attention. In: ICML (2020)
42. Tran, D., Wang, H., Torresani, L., Feiszli, M.: Video classification with channel-separated convolutional networks. In: ICCV (2019)
43. Tsai, Y.H.H., Bai, S., Yamada, M., Morency, L.P., Salakhutdinov, R.: Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In: EMNLP (2019)
44. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: NeurIPS (2017)
45. Vondrick, C., Pirsiavash, H., Torralba, A.: Anticipating visual representations from unlabeled video. In: CVPR (2016)
46. Wang, H., Zhu, Y., Green, B., Adam, H., Yuille, A., Chen, L.C.: Axial-deeplab: Stand-alone axial-attention for panoptic segmentation. In: ECCV (2020)

47. Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X., Van Gool, L.: Temporal segment networks for action recognition in videos. T-PAMI (2018)
48. Wang, S., Li, B.Z., Khabsa, M., Fang, H., Ma, H.: Linformer: Self-attention with linear complexity. arXiv preprint arXiv:2006.04768 (2020)
49. Wang, X., Zhang, S., Qing, Z., Shao, Y., Zuo, Z., Gao, C., Sang, N.: OadTR: Online action detection with transformers. In: ICCV (2021)
50. Werbos, P.J.: Backpropagation through time: what it does and how to do it. Proceedings of the IEEE 78(10), 1550–1560 (1990)
51. Wu, C.Y., Zaheer, M., Hu, H., Manmatha, R., Smola, A.J., Krähenbühl, P.: Compressed video action recognition. In: CVPR (2018)
52. Wu, C., Wang, Y., Shi, Y., Yeh, C.F., Zhang, F.: Streaming transformer-based acoustic models using self-attention with augmented memory. In: Interspeech (2020)
53. Xu, M., Gao, M., Chen, Y.T., Davis, L.S., Crandall, D.J.: Temporal recurrent networks for online action detection. In: ICCV (2019)
54. Xu, M., Xiong, Y., Chen, H., Li, X., Xia, W., Tu, Z., Soatto, S.: Long short-term transformer for online action detection. In: NeurIPS (2021)
55. Yun, S., Han, D., Oh, S.J., Chun, S., Choe, J., Yoo, Y.: Cutmix: Regularization strategy to train strong classifiers with localizable features. In: ICCV (2019)
56. Zach, C., Pock, T., Bischof, H.: A duality based approach for realtime tv-l 1 optical flow. In: Joint pattern recognition symposium. pp. 214–223. Springer (2007)
57. Zaheer, M., Guruganesh, G., Dubey, K.A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al.: Big bird: Transformers for longer sequences. In: NeurIPS. vol. 33 (2020)
58. Zeng, K.H., Shen, W.B., Huang, D.A., Sun, M., Carlos Niebles, J.: Visual forecasting by imitating dynamics in natural sequences. In: ICCV (2017)
59. Zhang, B., Wang, L., Wang, Z., Qiao, Y., Wang, H.: Real-time action recognition with enhanced motion vector cnns. In: CVPR (2016)
60. Zhang, H., Cisse, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. In: ICLR (2018)
61. Zhang, Q., Lu, H., Sak, H., Tripathi, A., McDermott, E., Koo, S., Kumar, S.: Transformer transducer: A streamable speech recognition model with transformer encoders and rnn-t loss. In: ICASSP (2020)