Streaming Dense Video Captioning

Xingyi Zhou* Anurag Arnab* Shyamal Buch Shen Yan
Austin Myers Xuehan Xiong Arsha Nagrani Cordelia Schmid
Google

Abstract

An ideal model for dense video captioning – predicting captions localized temporally in a video – should be able to handle long input videos, predict rich, detailed textual descriptions, and be able to produce outputs before processing the entire video. Current state-of-the-art models, however, process a fixed number of downsampled frames, and make a single full prediction after seeing the whole video. We propose a streaming dense video captioning model that consists of two novel components: First, we propose a new memory module, based on clustering incoming tokens, which can handle arbitrarily long videos as the memory is of a fixed size. Second, we develop a streaming decoding algorithm that enables our model to make predictions before the entire video has been processed. Our model achieves this streaming ability, and significantly improves the state-of-the-art on three dense video captioning benchmarks: ActivityNet, YouCook2 and ViTT. Our code is released at https://github.com/google-research/scenic.

1. Introduction

Video is ubiquitous in modern society, quickly becoming one of the most prevalent media formats for transmitting information. The majority of computer vision models designed for video understanding only process a handful of frames, mostly covering only a few seconds [28, 30, 36, 46, 55], and are typically limited to classifying these short segments into a fixed number of concepts. In order to achieve a comprehensive, fine-grained video understanding, we study the task of dense video captioning – jointly localizing events temporally in video and generating captions for them. Ideal models for this goal should be able to handle both long input sequences – to reason over long, untrimmed videos – and also to handle long output sequences in text space, to describe in detail all the events within the video.

Prior work on dense video captioning does not handle either long inputs or long outputs. Given a video of

---
*Equal contribution. {zhouxy, aarnab}@google.com

(a) **Conventional global dense video captioning models** [48, 56].
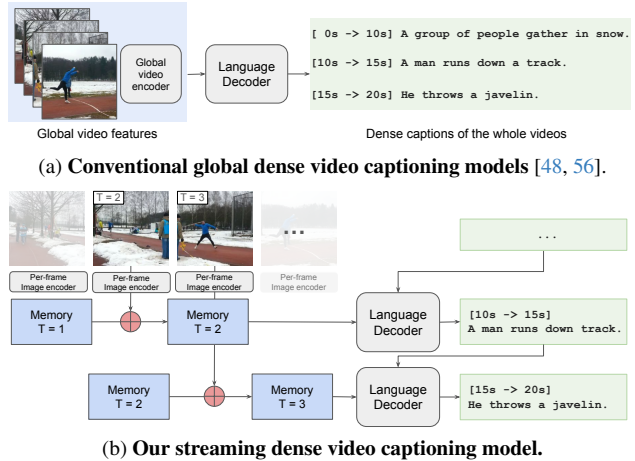

(b) **Our streaming dense video captioning model.**

Figure 1. **Comparing our streaming model (b) to conventional global models (a).** Conventional global models encode the entire video at once, and produce captions for all events at the end. Our model encodes images frame-by-frame, uses them to update a running memory, and predicts captions sequentially.

any length, state-of-the-art models either sample very few frames (e.g., 6 frames [45]) with large strides [10, 27, 45] (i.e., temporal downsampling), or keep one feature per-frame for all the frames [48, 56, 62] (i.e., spatial downsampling). With long textual output, current models solely rely on auto-regressive decoding [17] to generate multiple sentences at the end.

In this work, we design a *streaming* model for dense video captioning as shown in Fig. 1. Our streaming model does not require access to all input frames concurrently in order to process the video thanks to a memory mechanism. Moreover, our model can produce outputs causally without processing the entire input sequence, thanks to a new streaming decoding algorithm. Streaming models such as ours are inherently suited to processing long videos – as they ingest frames one at a time. Moreover, as the output is streamed, intermediate predictions are produced before processing the full video. This property means that streaming models can in theory be applied to process live video streams, as required for applications such as video conferencing, security and continuous monitoring among others.

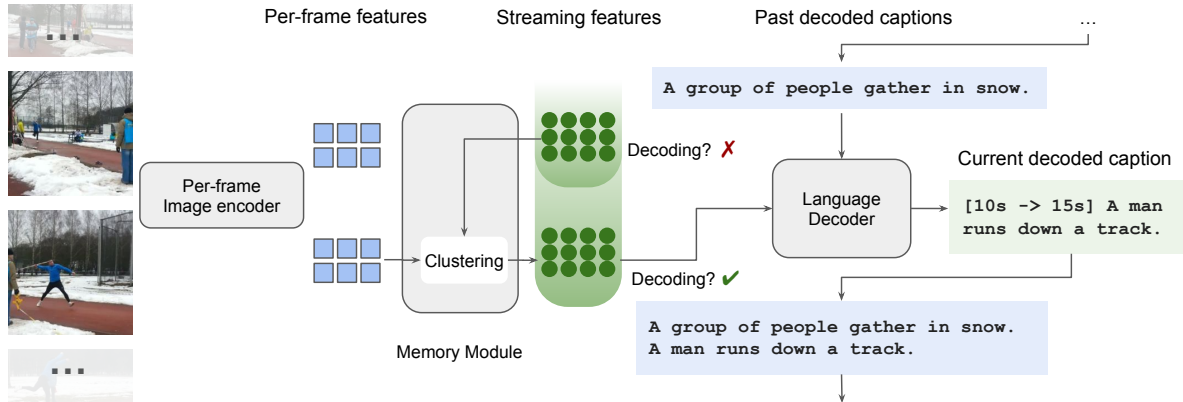In order to develop our streaming model, we first propose

Figure 2. **Illustration of our framework.** Each frame is passed through an image encoder, one at a time. A memory model, based on clustering, maintains compressed visual features from the beginning up to the current frame. At certain frames, denoted as "decoding points", we decode the representations from our memory into captions and their timestamps. Earlier text predictions, if available, are also passed as a prefix to the language decoder for the following decoding points. Our model can run on videos of arbitrary length, as the memory has a constant size, and can also output predictions before processing the whole video.

a novel memory mechanism, that takes frames once at a time. The memory model is based on K-means clustering, and uses a fixed number of cluster-center tokens to represent the video at each timestamp. We show that this method is simple and effective, and can process variable numbers of frames, with a fixed computational budget at decoding.

We also develop a streaming decoding algorithm, and train our network such that given a "decoding point" (Fig. 2) at a particular timestamp, it predicts all event captions that ended before it given the memory features at that timestamp. Our network is thus trained to make predictions at any timestamp of the video, and not just at the end of the video as in conventional, non-streaming models. Furthermore, we provide predictions from earlier decoding points as contexts for later ones. This context avoids predicting duplicated events, and can be used as an "explicit" memory in natural language summarising the earlier video. Our streaming output is also motivated by the fact that as the video length grows, our memory will inevitably lose information over time as its size is bounded. We avert this issue by making predictions before we have processed the entire video, and still keep early information via language context.

We evaluate our approach on three popular dense video captioning datasets, ActivityNet [26], YouCook2 [61] and ViTT [21]. Our results show that our streaming model significantly improves over the state-of-the-art, which inevitably uses fewer frames or fewer features, by up to **11.0** CIDEr points. We show that our method generalizes across both GIT [45] and Vid2Seq [56] architectures. Finally, our proposed memory can also be applied to paragraph captioning, improving baselines by 1-5 CIDEr points.

2. Related Work

Dense video captioning. Dense video captioning requires captioning events, and localizing them temporally. Tra-

ditionally, prior work used a two-stage approach, first localizing events in video, and then subsequently captioning them [22, 23, 26, 44, 47]. More recent end-to-end approaches include PDVC [63] which infers event captions and timestamps using a DETR-like [9] model. Vid2Seq [56] augments the vocabulary of a language model with timestamp tokens, allowing them to generate concatenated event captions in the same manner as a regular captioning model. We also use the output formulation of [56], as it integrates well with foundation vision-language models [45]. A related, but different problem, is that of audio description in movies [18, 19, 40], which requires generating captions for the visually impaired that must be complementary to speech, and often uses auxiliary, non-causal models to recognize characters or their speech [19].

As far as we are aware, all prior dense captioning models are not causal, as they encode the entire video at once. Moreover, to process long videos, they typically use visual features that are downsampled heavily (by selecting a few frames [10, 27, 45], or spatially pooling features per frame [48, 56, 62]). In contrast, we process videos in a *streaming* manner, processing long input sequences one frame at a time with a memory module, and streaming output sentences with a novel decoding algorithm.

Models for long videos. A common way of processing longer videos is to use a memory mechanism to provide a compact representation of past events. With transformers, memory can easily implemented by using tokens from past observations as inputs for the present time step [12, 33, 53]. Examples in vision include [20, 32, 50, 51] which preextract features offline and retrieve them during inference, and are thus not causal. MemViT [52] uses token activations from previous time steps as inputs to the current time step. However, this means that the sequence length grows over time and so it cannot handle arbitrarily long videos.

An alternate view of memory is to see it as a method of compressing previously observed tokens into a smaller, fixed-size set to be used at future time steps. Token Turing Machines [38] summarize past and current observations using the token summarization module of [37]. MovieChat [41] follows a similar idea, but uses a variant of Token Merging [5] instead to perform the summarization. TeSTra [59] uses an exponential moving average to integrate video features instead. The advantage of such approaches is that the memory bank has a fixed size, and therefore the computational cost remains bounded regardless of the length of the video. Our memory model has this same desirable property. However, our memory is based on clustering, using the centers from a K-means-like algorithm to summarize tokens from each time step, and we show experimentally that this outperforms other alternatives.

Causal models in video. Our streaming model is causal, meaning that its output only depends on current and past frames, without access to future frames. Although we are not aware of prior causal models for dense video captioning, there are causal models in many other vision domains. Online action detection [13, 25, 59, 60] aims to predict action labels for videos in real-time without access to future frames. Analogously, online temporal action localization [6, 24, 39] models also predict start- and end-times after an action is observed. Most models for object tracking [3, 49] and video object/instance segmentation [8, 29, 31] are causal too.

A common theme in the above tasks is that the model must make a prediction at each frame of the video. In contrast, we focus on dense video captioning [26], which is challenging as the output captions do not have a one-to-one correspondence with the video frames. We address this problem by proposing a streaming decoding algorithm.

3. Streaming Dense Video Captioning

Given a video $\mathbf{V} \in \mathbb{R}^{T \times H \times W \times 3}$, our goal is to produce a set of temporally localized captions: $\{(s, e, \mathbf{c})_1, \ldots, (s, e, \mathbf{c})_{n_e}\}$, where $s \in \mathbb{R}$ and $e \in \mathbb{R}$ are the starting and ending timestamps ($0 \leq s < e \leq T$), respectively, $\mathbf{c} = [w_1, \cdots, w_n]$ is a sequence of word tokens, and n_e the number of events. Each word token w_i is an integer in the range $[0, |V|]$, indexing the vocabulary V. We begin by describing conventional captioning models (Sec. 3.1), before detailing how we develop a streaming model (Fig. 2) by streaming inputs with memory (Sec. 3.2) and outputs with decoding points (Sec. 3.3).

3.1. Preliminaries

Captioning models broadly consist of a vision encoder followed by a text decoder. We outline these approaches, and show how they can be extended to dense captioning next.

Visual encoder. The first step is to encode the video into features $\mathbf{f} = \mathcal{F}(\mathbf{V}), \mathbf{f} \in \mathbb{R}^{N \times D}$, where N is the feature resolution (*i.e.*, number of tokens for transformer-based encoders), and D is the feature dimension. The visual feature encoder $\mathcal{F}$ can be a native video backbone [1, 4], or an image encoder [34, 57] applied to each individual frame. In the latter case, the video feature is a stack of image features, *i.e.* $N = T \cdot N_f$, where N_f is the number of tokens per frame. We use a per-frame encoding, but instead of pre-extracting them from the whole video, we use a memory mechanism to process the features in a causal, streaming fashion (Sec. 3.2) that can generalize to longer video durations.

Text decoder. Given the visual features, $\mathbf{f}$, and optional textual prefix tokens, $\mathbf{p}$, the text decoder, $\mathcal{D}$ generates a sequence of word tokens, $\mathbf{c}$ from them. We use an autoregressive [17, 42] decoder that generates the next word token, w_i, conditioned on previous words, $\mathbf{w}_{1:i-1}$, and prefix if provided as $w_i = \mathcal{D}(\mathbf{f}, \mathbf{p}, \mathbf{w}_{1:i-1})$. Note that prefix tokens are typically not used in captioning tasks, but are used in question-answering (QA) to encode the input question. Concretely, the text decoder, $\mathcal{D}$, is a sequence of transformer layers [42] operating on a concatenation of visual features $\mathbf{f}$ and word embeddings of the prefix [35, 45]. This architecture is shown to be effective in both captioning and QA tasks across image and video [10, 27, 45].

Dense video captioning with timestamps. Combining the above visual encoder and text decoder gives a basic architecture for video captioning. To extend it for captioning multiple events with starting and ending timestamps, Vid2Seq [56] introduced two main modifications: First, it augments the vocabulary, V', of the captioning model with time tokens, w^s and w^e, which represent the starting and ending times, respectively. A single event is therefore represented as $\mathbf{c}' = [w^s, w^e, w_1, \cdots, w_n]$, and $|V'| = |V| + |T|$ where $|V| \leq w^s < w^e \leq |V'|$, and $|T|$ is the number of time tokens. Second, Vid2Seq concatenates all timed captions into a single long caption that is ordered by starting time: $\mathbf{C} = [\mathbf{c}'_1, \mathbf{c}'_2, \cdots, \mathbf{c}'_{n_e}]$ where n_e is the number of events. Therefore, dense video captioning can be formulated as standard video captioning with target $\mathbf{C}$.

Despite its effectiveness, the (generalized) Vid2Seq [56] architecture has a number of key limitations: First, it forwards visual features from the whole video, $\mathbf{f}$ through the decoder, meaning that it does not scale effectively to longer videos and more tokens. In addition, as Vid2Seq predicts all event captions at once, after processing the whole video, it struggles with predicting long, detailed captions. To address these issues, we introduce **streaming** dense video captioning models, where we process the inputs once at a time using a memory module to bound computational costs, and stream the outputs such that we can make predictions before processing the whole video.

Algorithm 1: Updating memory tokens at a timestamp.

Input	: $\mathbf{M}_{t-1} \in \mathbb{R}^{K \times D}$: memory tokens
	$\mathbf{W}_{t-1} \in \mathbb{R}^{K}$: weights of memory tokens
	$\mathbf{f}_t \in \mathbb{R}^{N_f \times D}$: incoming tokens
Hyperparameters:	τ: number of K-means iterations.
Output	: $\mathbf{M}_t \in \mathbb{R}^{K \times D}$: updated memory tokens
	$\mathbf{W}_t \in \mathbb{R}^{K}$: updated weights of memory

1 $\mathbf{X} \leftarrow [\mathbf{M}_{t-1}, \mathbf{f}_t]$ // Concatenate memory and incoming tokens.
2 $\mathbf{W} \leftarrow [\mathbf{W}_{t-1}, \mathbf{1}]$ // Initialize incoming weights and concatenate.
3 $\mathbf{M}_t \leftarrow \mathbf{M}_{t-1}$ // Initialize new cluster centers as the old centers.
4 **for** $i \leftarrow 1$: τ **do**
5      $\mathbf{d} \leftarrow$ pairwise_l2_distance$(\mathbf{X}, \mathbf{M_t})$ // Shape $(K+N_f, K)$.
6      $\boldsymbol{\delta} \leftarrow \mathbf{d}$.argmin(axis=1)   // Assign each token to a center.
7      $\boldsymbol{\delta} \leftarrow$ make_onehot$(\boldsymbol{\delta}, K)$ // Binary. Shape$(K+N_f, K)$.
8      $\mathbf{W}_t \leftarrow \boldsymbol{\delta}^\top \mathbf{W}_t$ // Compute #tokens assigned to each center.
9      $\mathbf{A} \leftarrow \boldsymbol{\delta}^\top / \mathbf{W}_t$ // Weight matrix. "/" is elementwise div.
10     $\mathbf{M}_t \leftarrow \mathbf{A}\mathbf{X}$ // Compute new centers as a linear function.
11 **end**
12 **return** $\mathbf{M}_t, \mathbf{W}_t$

3.2. Streaming inputs using memory

The input visual features, $\mathbf{f}$, have dimensionality $\mathbb{R}^{T \cdot N_f \times D}$, where typical values are $T > 64$ for a sparsely sampled (*e.g.* 1 FPS), few-minute-long video, and $N_f = 257$ tokens per-frame for a vision transformer such as CLIP [34]. Directly feeding all $T \cdot N_f$ to the text decoder is prohibitively expensive, due to the quadratic complexity of self-attention. Therefore, existing methods aggressively downsample $\mathbf{f}$ to reduce the number of tokens (by temporally sampling a few frames with large strides [10, 45], or spatially subsampling each frame to a single token [48, 56, 62]). Even so, we will reach memory limits with longer videos, and the information required for fine-grained localization and description is lost. Therefore, rather than aggressive downsampling, we use a memory mechanism to process all tokens frame-by-frame, which ensures that the computational cost is bounded irrespective of the length of the video.

Let K be a pre-defined memory size, the memory at each time $\mathbf{M}_t$ would always be the constant size K, i.e., $\mathbf{M}_t \in \mathbb{R}^{K \times D}, \forall t$. We interpret $\mathbf{M}$ as being a summary of all relevant information in the video, and initialize it by taking the first K tokens from $\mathbf{f}$. Therefore, we set K as a multiple of N_f, such that the initial memory is the features of the first $\frac{K}{N_f}$ frames: $\mathbf{M}_{K/N_f} = [\mathbf{f}_1, \cdots, \mathbf{f}_{K/N_f}]$.

Next, we update the memory at each timestamp for each incoming frame $\mathbf{f}_t$. Our intuition is to keep as much *diverse* information in the original video as possible, while not increasing the storage budget (*i.e.* by keeping a constant memory size K). We thus propose a K-means-like clustering algorithm, to use the feature cluster centers as the approximate video features. To avoid the cluster centers biasing quickly to incoming features, we keep track of the number of merged tokens in each cluster center. We use this as a momentum weight, so that cluster centers that are merged from more tokens change slower. The detailed algorithm diagram is provided in Alg. 1, and illustrated in Fig. 3.
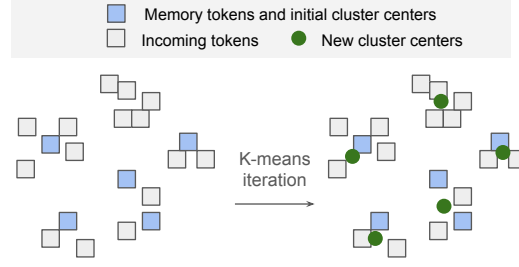

Figure 3. **Illustration of our clustering-based memory module.** The current memory tokens are shown by blue squares. At each time step, the memory tokens evolve by integrating information from the incoming tokens (gray squares), using K-means iterations to produce the updated memory tokens (green circles).

The K-means algorithm is not differentiable with respect to the assignment of data points to cluster centers, $\boldsymbol{\delta}$ (Line 6 of Alg.1). However, the inputs and outputs of our memory module are the updated cluster centers, $\mathbf{M}_t$, which is a linear mapping of the input $\mathbf{X} = [\mathbf{M}_{t-1}, \mathbf{f}_t]$, as $\mathbf{M}_t = \mathbf{A}\mathbf{X}$, where $\mathbf{A}$ is a weight matrix computed from $\mathbf{X}$. Therefore, even though we cannot compute the gradient of $\mathbf{A}$ with respect to $\mathbf{X}$, we can compute the gradient of $\mathbf{M}_t$ with respect to the input $\mathbf{X}$, and thus to the input visual feature $\mathbf{f}$. As a result, we can use our memory module in any part of a neural network, and learn parameters in preceding layers.

3.3. Streaming outputs with decoding points

The memory module from Sec. 3.2 enables us to efficiently ingest long input videos. However, it is still desirable for our model's text decoder to predict outputs before it has processed the entire input sequence: Streaming the output substantially decreases the latency of the model, as we do not have to wait for the model to process the entire input sequence to make predictions. This is particularly relevant for processing, for example, live video streams. Furthermore, streaming the output can in fact increase our model's accuracy: As we have a memory with a fixed size, K, from which we decode outputs, we will inevitably lose information over time. We can therefore avert this issue by making predictions before we have processed the entire video.

Decoding points   As shown in Fig. 4, we define "decoding points", d_i, as intermediate timestamps where we decode event captions given the features in our memory, $\mathbf{M}_{d_i}$. We train our model such that at each decoding point, d_i, the model predicts all event captions that finished before it. More specifically,

$$\mathcal{Y}_i = \{(w_j^s, w_j^e, \mathbf{c}_j) | w_j^e \leq d_i\}, \tag{1}$$

where $\mathcal{Y}_i$ is the set of all event captions corresponding to the ith decoding point d_i, and w_j^s, w_j^e are the starting and ending time of the jth event.
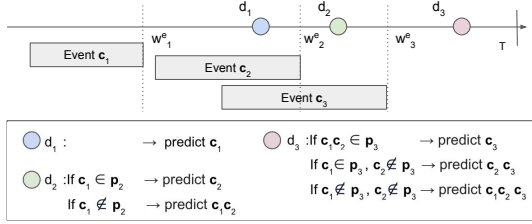
Figure 4. Decoding point supervision in training. A decoding point, d_i, can be at any frame. At each point, we take the memory features, $\mathbf{M}_{d_i}$, and predict all events that have finished before d_i, and are not in the prefix $\mathbf{p}$. Therefore, the union between the prefix and the prediction target covers all events finished before it.

As decoding points are applied sequentially, later decoding points should have access to the predictions of earlier decoding points, and should not repeat them again. Therefore, from the second decoding point onwards, we concatenate the outputs of previous decoding points as the prefix to the text decoder, as shown in Fig. 4. Moreover, during training, we perform further data augmentation by randomly removing some of the previous event captions from the prefix, and adding them to the target instead, to increase robustness to potential errors in earlier predictions. We therefore denote our prefixes and captioning targets during training as

$$\mathbf{p}_i = [\mathbf{c}'_1, \mathbf{c}'_2, \ldots, \mathbf{c}'_{j-1}] \quad (2)$$

$$\mathbf{y}_i = [\mathbf{c}'_j, \mathbf{c}'_{j+1}, \ldots, \mathbf{c}'_{|\mathcal{Y}_i|}], \quad (3)$$

where $j < |\mathcal{Y}_i|$ is a randomly chosen splitting point to partition the target and context. During inference, we use the models actual predictions, $\mathbf{p}_i = [\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_{i-1}]$, instead.

In practice, we uniformly sample decoding points during both training and inference, with a stride of S frames. Since the model is trained to predict all event captions before the decoding point, it means that the exact location at inference time does not need to match the event boundaries closely. The number of decoding points can be also different between training and inference, as we will ablate. This method is both simple and scalable, as we show experimentally in the next section.

4. Experimental Evaluation

4.1. Experimental Setup

4.1.1 Datasets and Evaluation Metrics

We evaluate our model on the three most popular dense video captioning datasets: ActivityNet Captions [26], YouCook2 [61], and ViTT [21].

ActivityNet Captions [26] contains 8,649 training videos and 4,267 validation videos (considering videos that are still online on YouTube). The videos are untrimmed, each containing multiple action events, with an average video length of 2 minutes and an average of 3.7 events. The videos are selected from the ActivityNet [7], which covers a wide range of visual domains. Each video is carefully labeled by human annotators. We use this dataset for both dense captioning (predicting a sequence of captions with their start- and end-times) and paragraph captioning (predicting the aforementioned captions as a single paragraph without timestamps).

YouCook2 [61] contains 1,333 training videos and 456 validation videos. The videos are on average 5.3 minutes long, with an average of 7.8 events. This dataset focuses on cooking scenes, and most videos contain one main actor cooking while describing the recipe. The videos are manually annotated, aided by speech transcriptions obtained through Automatic Speech Recognition (ASR). As expected due to the annotation process, using ASR as an additional modality can be used to improve model predictions [56]. However, our work focuses on using visual-only inputs, to study our proposed streaming model. We also perform dense and paragraph captioning on this dataset.

ViTT [21] contains 4,608 training videos and 2,301 testing videos, which are on average 4.7 minutes long and contain 7 events per video. It is collected in a similar way as YouCook2, but focuses on more diverse domains. Again, we do not use the ASR inputs in our experiments.

Evaluation Metrics For all datasets, we report standard dense captioning evaluation metrics following Vid2Seq [56] and PDVC [48]. The primary metric is CIDEr which has been adapted to jointly evaluate captioning and localisation. In particular, we average the CIDEr score [43] for positive ground-truth-prediction pairs with a temporal IoU above the thresholds, $\{0.3, 0.5, 0.7, 0.9\}$. Similarly, we report METEOR [2] averaged over multiple IoU thresholds. Finally, we report the recently proposed $SODA_c$ [16], which considers all event captions from the same video, and is therefore a more holistic measure.

4.1.2 Implementation details

We implement our streaming model on two video captioning architectures, GIT [45] and Vid2Seq [56]. Both use a ViT-L [14] initialized from CLIP [34] as image encoder $\mathcal{F}$.

GIT [45] concatenates all tokens from all input frames, $\mathbf{f}$, and feeds them to a 6-layer transformer language decoder. We apply our streaming input module before the language decoder, *i.e.*, rather than concatenating frame features, we use our memory features instead. The original GIT paper pretrains the language decoder on in-house data [45]. As we do not have access to the original data or weights, we pretrain our GIT model on the WebLI dataset [11].

Vid2Seq [56] pools all frame tokens spatially into 1 token per frame, and then applies a 12-layer temporal transformer [1] to obtain the final visual features. The language decoder is a 12-layer T5-Base decoder [35]. Vid2Seq pre-

	#Tokens	$T{=}16$	$T{=}32$	$T{=}64$	$T{=}128$
No memory	$T{\times}N_f$	**29.8**	-	-	-
Spatial pooling	T	27.6	27.3	27.9	27.4
Temporal pooling	N_f	29.3	28.0	26.8	25.2
EMA [59]	N_f	28.2	26.3	22.0	16.3
MovieChat [41]	K	29.3	**29.2**	28.9	29.3
Clustering (ours)	K	**29.8**	**29.2**	**30.6**	**30.4**

Table 1. **Ablation on memory modules.** We show CIDEr (averaged over multiple IoUs) on ActivityNet with GIT under different # input frames T. The 2nd column shows the number of input tokens to the language decoder. $N_f{=}257$ is the number of tokens per-frame. $K{=}514$ is the number of memory tokens. Our module benefits from more frames, and outperforms other alternatives.

K	CIDEr	Iters.	CIDEr	Momentum	CIDEr
257	29.4	1	30.3	✗	29.7
257×2	**30.6**	2	**30.6**	✓	**30.6**
257×3	29.8	4	30.3		

(a) #memory tokens. (b) K-means iterations. (c) Momentum term in clustering

Table 2. **Ablation on memory module hyperparameters.** We perform experiments on ActivityNet with GIT with 64 frames.

trains the temporal transformer and the T5 decoder on narrated videos from the YT-Temporal dataset [58]. We use their public, pretrained weights as our initialization. We apply our streaming input module before the temporal transformer, to ensure the intermediate features remain causal.

When finetuning our model for each dataset, we freeze the frame encoder following prior works [48, 56, 63]. We provide detailed training hyperparameters in the supplementary. For each architecture, the training hyperparameters are shared across the 3 datasets.

4.2. Analysis of Streaming Modules

We first analyze each of the main components of our model: streaming the input with our memory module (Sec. 3.2), and then streaming outputs using decoding points (Sec. 3.3). Unless otherwise stated, we use the GIT [45] backbone for our experiments. Due to randomness in training and evaluation, for all ablation experiments (Tab. 1-Tab. 3), we repeat the same run 3 times and report the averaged metrics. When compared to other methods (Tab. 4, Tab. 5), we report results for the best model.

4.2.1 Streaming inputs with memory

As the GIT [45] architecture concatenates visual features, $\mathbf{f}$, from multiple frames before passing it to the language decoder, it is limited by the number of frames that it can process. Our clustering-based memory module allows us

Number of decoding points	1	2	4	8	16	20
CIDEr		30.6	34.5	38.8	39.5	**40.6**	40.4

(a) Number of decoding points during training.

Prefix	CIDEr	Aug. Prefix	CIDEr	Stride	CIDEr
None	23.1	✗	37.6	32	40.6
Captions	**40.6**	✓	40.6	21	34.7
Captions & time	39.3				

(b) Context provided after the first decoding point. (c) Random masking prefix as augmentation during training. (d) Decoding point stride during inference (input 64 frames).

Table 3. **Ablation on streaming outputs.** (a) Increasing the number of decoding points during training consistently improves accuracy, as it provides more supervision to the model. (b) It is critical to provide context after the first decoding point, so that the model does not repeat predictions. Providing captions alone, without timestamps is sufficient. (c) It is important to provide imperfect prefixes during training, to mimic model behavior in inference. (d) Stride used for decoding during inference.

to process more frames, and we consider the following approaches to evaluate it:

No memory: We simply concatenate all visual tokens from all frames as done in GIT [45]. Due to memory constraints, we can only feed up to 16 frames to the decoder.

Spatial- or temporal-pooling: We pool the visual features, $\mathbf{f}$, along either the spatial or temporal dimensions to reduce the number of tokens fed to the language decoder.

EMA: We use an exponential moving average of frame features, $\mathbf{f}_t$, at each time step, following TeSTra [59]. We sweep the decay rate from $\{0.9, 0.99, 0.999\}$, finding 0.9 to perform best.

MovieChat [41]. Finally, the recent MovieChat paper maintains a memory of K tokens. For each incoming frame, it sequentially processes each token, and merges the two most similar tokens in the memory bank such that the size remains fixed at K. We implement this method using the author's public code.

Tab. 1 compares the results of the different memory modules. For $T{=}16$, where we can feed all the tokens from the vision backbone, $\mathbf{f}$, into the decoder, "no memory" and our method both performs the best. We expected "no memory" to perform well, as it uses the most tokens, $T{\times}N_f$. However, our clustering-based method achieves the same performance, which suggests that it is able to effectively capture the relevant information in the video with $8\times$ fewer tokens, and is causal too. It is not possible to use "no memory" for $T{>}16$ due to its computational cost.

With more frames, naïvely pooling along the spatial-, or temporal-dimensions actually performs worse. This is likely because we are averaging out information over longer temporal durations, and thus losing the details required for more detailed localization or captioning. Our method and MovieChat on the other hand, are able to leverage more frames to improve performance, as they keep diverse fea-

	ActivityNet				YouCook2				ViTT			
	CIDEr	SODA	Meteor	F1	CIDEr	SODA	Meteor	F1	CIDEr	SODA	Meteor	F1
MT [62]	9.3	–	5.0	–	6.1	–	3.2	–	–	–	–	–
E2ESG [63]	–	–	–	–	25.0	–	3.5	–	–	–	–	–
PDVC [48]	29.3	6.0	7.6	–	28.9	4.9	5.7	–	–	–	–	–
GIT [45]	29.8	5.7	7.8	50.6	12.1	3.1	3.4	17.7	15.1	7.1	3.4	32.5
Vid2Seq† [56]	30.2	5.9	8.5	51.8	25.3	5.7	6.4	23.5	23.0	9.8	5.0	**37.7**
Streaming GIT (ours)	**41.2**	**6.6**	9.0	50.9	15.4	3.2	3.6	16.6	18.5	8.3	4.0	33.9
Streaming Vid2Seq (ours)	37.8	6.2	**10.0**	**52.9**	**32.9**	**6.0**	**7.1**	**24.1**	25.2	**10.0**	**5.8**	35.4

Table 4. **Comparison to the state-of-the-art on dense video captioning** We add our streaming model to both GIT [45] and Vid2Seq [56], denoted by Streaming GIT and Streaming Vid2Seq, respectively, achieving consistent and substantial improvements across three datasets. All models use only visual inputs. †denotes version with visual-only inputs.

tures within the memory. Finally, our clustering method outperforms MovieChat and other memory modules for all numbers of frames that we considered, which is why we use it for all future experiments.

We also ablate the hyperparameters of our memory module in Tab. 2. Following this experiment, we set $K = 257 \times 2$, or the number of tokens in two frames, and use 2 iterations of K-means, as it achieves the best performance. Our proposed momentum term (Sec. 3.2), which prevents cluster centers from becoming too biased towards incoming frame tokens, also improves CIDEr from 29.7 to 30.6.

4.2.2 Streaming outputs

We now analyze the effect of our streaming decoding method (Sec. 3.3), which enables us to predict event captions from intermediate features in our memory, and utilize previous prediction as the prefix to our text decoder. Table 3 analyses the effects of the number of decoding points during training, the impact of the prefix, and how we select decoding points during inference.

Table 3a shows that we achieve significant improvements by increasing the number of decoding points during training, improving the CIDEr score by 10 points, or 33% relative, compared to only making predictions at the end, as in conventional models. Streaming the output with decoding points can provide performance benefits for multiple reasons: First, as we have multiple decoding points, the required output caption is shorter at each decoding point, making the captioning task easier. Second, the visual features from our memory, **M**, may be aligned better with the target text, since **M** does not represent the visual features from the entire video as in baseline approaches, but only the features up to the decoding point. Finally, training with decoding points provides a stronger training signal to the model, as we provide training supervision at each decoding point. Moreover, it should aid generalization, as the network is being trained to make consistent predictions from more points along the timeline.

Tab. 3b shows that it is essential to provide past predic-

tions as the prefix. Without a prefix, the model is poor, underperforming our non-streaming baseline of only decoding at the end (30.6). This is because the model predicts duplicated predictions at each decoding point. Providing past captions outperforms this baseline substantially. We find that also adding previously predicted timestamps, does not improve over captions alone, suggesting that temporal boundaries of past events are not particularly informative for future events. Tab. 3c further shows that while training with a prefix, it is important to mimic inference behavior by including missing captions in earlier predictions.

Table 3d examines the choice of decoding points during inference. We find that using a stride, $S = 32$ (which equates to a decoding point at the middle and end of a 64-frame clip), performs considerably better than $S = 21$ (three uniformly chosen points). We observed qualitatively that using too many decoding points during inference can sometimes still result in the model making duplicate predictions (even with past prediction as the prefix). Whilst it is also possible to remove duplicate predictions with non-maximal suppression (NMS [15]), it is more challenging for captioning models as we also require a calibrated score for each event caption to perform NMS. We therefore leave an investigation of NMS for dense video captioning to future work.

4.2.3 Generalization to backbones and datasets

To show the generality of our method, we add our streaming modules onto both the GIT [45] and Vid2Seq [56] architectures, which we denote as Streaming GIT and Streaming Vid2Seq, respectively.

The last two rows of Tab. 4 shows that we improve substantially over both baselines consistently on three datasets. Our GIT baseline can process a maximum of $N_f = 16$ frames due to memory limitations, and our improvement also stems from the fact that we use $N_f = 64$ for our Streaming GIT thanks to our memory module. Vid2Seq pools visual tokens spatially such that it only uses a single token per frame. Therefore, our Streaming Vid2Seq does not use more frames than the baseline, and the improvement is due
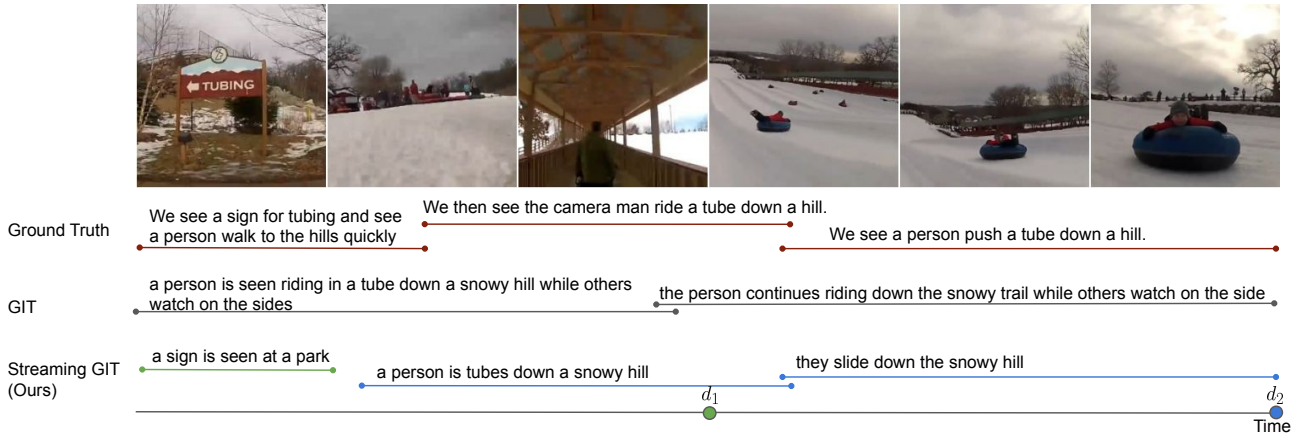
Figure 5. **Qualitative results on ActivityNet validation.** Results from the ground truth (top), the baseline (middle), and our model (bottom). We show outputs from two decoding points in green and blue respectively. Our model captures more details than the baseline.

to our streaming of output event captions, which improves performance significantly as shown in Tab. 3.

Finally, we observe that our Vid2Seq baseline performs substantially better than the GIT baseline on YouCook2. This difference is due to the pretraining: We used the public, pretrained Vid2Seq checkpoint [56] which was pretrained on YT-Temporal [58] – a dataset with a similar domain to YouCook2. Note that this is the same experimental protocol as Vid2Seq [56], the current state-of-the-art.

4.3. State-of-the-art Comparison

Tab. 4 also compares our method to the state-of-the-art among dense video captioning methods using only video frames as inputs. We achieved substantial gains over prior, published works, notably improving CIDEr on ActivityNet by 11.0 points, and YouCook2 by 4.0 points, respectively. We also achieved improvements, albeit smaller, on SODA and Meteor. Our improvements on localization (F1) are smaller, showing the gains are more from better captioning qualities. Fig. 5 visualizes an example on ActivityNet.

We note that it is possible to further improve results, particularly on YouCook2, by using Automatic Speech Recognition (ASR) as an additional input modality [56]. This is primarily because the spoken utterances of the actors are well aligned with the visual content, and because ASR was used in the annotation procedure of the dataset itself. However, integrating multiple modalities such as ASR is orthogonal to the main focus of this work.

Paragraph captioning. In addition to dense captioning, we also compare with state-of-the-art models on the same datasets for paragraph captioning, which aims to predict the captions throughout the entire video, but without any timestamps. Therefore, we only apply our streaming input model here, as the timestamps needed to assign decoding points during training are not available in this setting.

We train our Streaming GIT model for paragraph captioning, on both ActivityNet [26] and Youcook2 [61]. Tab. 5

	ActivityNet	YouCook2
MFT [54]	19.1	-
PDVC [48]	20.5	-
Vid2Seq [56]†	28.0	26.6
GIT [45]	32.5	28.4
Ours	**33.4**	**33.9**

Table 5. **State-of-the-art comparison on paragraph captioning.** We report CIDEr on ActivityNet and YouCook2 validation set. We compare models that do not require action proposal inputs and only takes visual inputs. †denotes version with visual-only inputs. Our model achieves the best performance on both datasets.

shows that we achieve state-of-the-art results on this task too. The GIT baseline is our model trained on the same setting as our full model, but uses 16 input frames with all tokens concatenated for the decoder. This baseline already outperforms the state-of-the-art from visual-only inputs. Our model uses more input frames (64 frames), and further boosts the performance by 0.9 and 5.5 points on the two datasets, respectively, showing the benefits of our memory module which is consistent with our results in Tab. 1.

5. Conclusion and Future Work

We have proposed a streaming model for dense video captioning with two novel components: A clustering-based memory that can efficiently handle arbitrarily long videos with bounded computation, and a streaming decoding algorithm that enables our model to make predictions before the entire video has been processed. We achieve this streaming ability while also improving the state-of-the-art on five dense- and paragraph-captioning tasks.

Future work is to develop a benchmark for dense video captioning which requires reasoning over longer videos than current datasets, to better evaluate the abilities of streaming models such as ours.

References

[1] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. ViViT: A Video Vision Transformer. In *ICCV*, 2021. 3, 5

[2] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *ACL Workshops*, 2005. 5

[3] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixe. Tracking without bells and whistles. In *ICCV*, pages 941–951, 2019. 3

[4] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *ICML*, 2021. 3

[5] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. In *ICLR*, 2023. 3

[6] Shyamal Buch, Victor Escorcia, Chuanqi Shen, Bernard Ghanem, and Juan Carlos Niebles. Sst: Single-stream temporal action proposals. In *CVPR*, 2017. 3

[7] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *CVPR*, 2015. 5

[8] Sergi Caelles, Jordi Pont-Tuset, Federico Perazzi, Alberto Montes, Kevis-Kokitsi Maninis, and Luc Van Gool. The 2019 davis challenge on vos: Unsupervised multi-object segmentation. In *arXiv:1905.00737*, 2019. 3

[9] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020. 2

[10] Xi Chen, Josip Djolonga, Piotr Padlewski, Basil Mustafa, Soravit Changpinyo, Jialin Wu, Carlos Riquelme Ruiz, Sebastian Goodman, Xiao Wang, Yi Tay, et al. Pali-x: On scaling up a multilingual vision and language model. In *arXiv:2305.18565*, 2023. 1, 2, 3, 4

[11] Xi Chen, Xiao Wang, Soravit Changpinyo, AJ Piergiovanni, Piotr Padlewski, Daniel Salz, Sebastian Goodman, Adam Grycner, Basil Mustafa, Lucas Beyer, et al. Pali: A jointly-scaled multilingual language-image model. In *ICLR*, 2023. 5

[12] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *ACL*, 2019. 2

[13] Roeland De Geest, Efstratios Gavves, Amir Ghodrati, Zhenyang Li, Cees Snoek, and Tinne Tuytelaars. Online action detection. In *ECCV*, 2016. 3

[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021. 5

[15] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *PAMI*, 2009. 7

[16] Soichiro Fujita, Tsutomu Hirao, Hidetaka Kamigaito, Manabu Okumura, and Masaaki Nagata. Soda: Story oriented dense video captioning evaluation framework. In *ECCV*, 2020. 5

[17] Alex Graves. Generating sequences with recurrent neural networks. In *arXiv:1308.0850*, 2013. 1, 3

[18] Tengda Han, Max Bain, Arsha Nagrani, Gül Varol, Weidi Xie, and Andrew Zisserman. Autoad: Movie description in context. In *CVPR*, 2023. 2

[19] Tengda Han, Max Bain, Arsha Nagrani, Gul Varol, Weidi Xie, and Andrew Zisserman. Autoad ii: The sequel-who, when, and what in movie audio description. In *ICCV*, 2023. 2

[20] Roei Herzig, Elad Ben-Avraham, Karttikeya Mangalam, Amir Bar, Gal Chechik, Anna Rohrbach, Trevor Darrell, and Amir Globerson. Object-region video transformers. In *CVPR*, 2022. 2

[21] Gabriel Huang, Bo Pang, Zhenhai Zhu, Clara Rivera, and Radu Soricut. Multimodal pretraining for dense video captioning. *arXiv:2011.11760*, 2020. 2, 5

[22] Vladimir Iashin and Esa Rahtu. A better use of audio-visual cues: Dense video captioning with bi-modal transformer. In *BMVC*, 2020. 2

[23] Vladimir Iashin and Esa Rahtu. Multi-modal dense video captioning. In *CVPR Workshops*, 2020. 2

[24] Hyolim Kang, Kyungmin Kim, Yumin Ko, and Seon Joo Kim. Cag-qil: Context-aware actionness grouping via q imitation learning for online temporal action localization. In *ICCV*, 2021. 3

[25] Dan Kondratyuk, Liangzhe Yuan, Yandong Li, Li Zhang, Mingxing Tan, Matthew Brown, and Boqing Gong. Movinets: Mobile video networks for efficient video recognition. In *CVPR*, 2021. 3

[26] Ranjay Krishna, Kenji Hata, Frederic Ren, Li Fei-Fei, and Juan Carlos Niebles. Dense-captioning events in videos. In *ICCV*, 2017. 2, 3, 5, 8

[27] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *ICML*, 2023. 1, 2, 3

[28] Kunchang Li, Yali Wang, Yinan He, Yizhuo Li, Yi Wang, Limin Wang, and Yu Qiao. Uniformerv2: Spatiotemporal learning by arming image vits with video uniformer. In *arXiv:2211.09552*, 2022. 1

[29] Mengtian Li, Yu-Xiong Wang, and Deva Ramanan. Towards streaming perception. In *ECCV*, 2020. 3

[30] Ziyi Lin, Shijie Geng, Renrui Zhang, Peng Gao, Gerard de Melo, Xiaogang Wang, Jifeng Dai, Yu Qiao, and Hongsheng Li. Frozen clip models are efficient video learners. In *ECCV*, 2022. 1

[31] Anton Milan, Laura Leal-Taixé, Ian Reid, Stefan Roth, and Konrad Schindler. Mot16: A benchmark for multi-object tracking. In *arXiv:1603.00831*, 2016. 3

[32] Junting Pan, Siyu Chen, Mike Zheng Shou, Yu Liu, Jing Shao, and Hongsheng Li. Actor-context-actor relation network for spatio-temporal action localization. In *CVPR*, 2021. 2

[33] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebas-

tian Riedel. Language models as knowledge bases? In *arXiv:1909.01066*, 2019. 2

[34] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021. 3, 4, 5

[35] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 2020. 3, 5

[36] Chaitanya Ryali, Yuan-Ting Hu, Daniel Bolya, Chen Wei, Haoqi Fan, Po-Yao Huang, Vaibhav Aggarwal, Arkabandhu Chowdhury, Omid Poursaeed, Judy Hoffman, et al. Hiera: A hierarchical vision transformer without the bells-and-whistles. In *arXiv:2306.00989*, 2023. 1

[37] Michael S Ryoo, AJ Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. Tokenlearner: What can 8 learned tokens do for images and videos? In *NeurIPS*, 2021. 3

[38] Michael S Ryoo, Keerthana Gopalakrishnan, Kumara Kahatapitiya, Ted Xiao, Kanishka Rao, Austin Stone, Yao Lu, Julian Ibarz, and Anurag Arnab. Token turing machines. In *CVPR*, 2023. 3

[39] Gurkirt Singh, Suman Saha, Michael Sapienza, Philip HS Torr, and Fabio Cuzzolin. Online real-time multiple spatiotemporal action localisation and prediction. In *ICCV*, 2017. 3

[40] Mattia Soldan, Alejandro Pardo, Juan León Alcázar, Fabian Caba, Chen Zhao, Silvio Giancola, and Bernard Ghanem. Mad: A scalable dataset for language grounding in videos from movie audio descriptions. In *CVPR*, 2022. 2

[41] Enxin Song, Wenhao Chai, Guanhong Wang, Yucheng Zhang, Haoyang Zhou, Feiyang Wu, Xun Guo, Tian Ye, Yan Lu, Jenq-Neng Hwang, et al. Moviechat: From dense token to sparse memory for long video understanding. In *arXiv:2307.16449*, 2023. 3, 6

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017. 3

[43] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. In *CVPR*, 2015. 5

[44] Jingwen Wang, Wenhao Jiang, Lin Ma, Wei Liu, and Yong Xu. Bidirectional attentive fusion with context gating for dense video captioning. In *CVPR*, 2018. 2

[45] Jianfeng Wang, Zhengyuan Yang, Xiaowei Hu, Linjie Li, Kevin Lin, Zhe Gan, Zicheng Liu, Ce Liu, and Lijuan Wang. Git: A generative image-to-text transformer for vision and language. In *arXiv:2205.14100*, 2022. 1, 2, 3, 4, 5, 6, 7, 8

[46] Limin Wang, Bingkun Huang, Zhiyu Zhao, Zhan Tong, Yinan He, Yi Wang, Yali Wang, and Yu Qiao. Videomae v2: Scaling video masked autoencoders with dual masking. In *arXiv:2303.16727*, 2023. 1

[47] Teng Wang, Huicheng Zheng, Mingjing Yu, Qian Tian, and Haifeng Hu. Event-centric hierarchical representation for dense video captioning. *IEEE Transactions on Circuits and Systems for Video Technology*, 2020. 2

[48] Teng Wang, Ruimao Zhang, Zhichao Lu, Feng Zheng, Ran Cheng, and Ping Luo. End-to-end dense video captioning with parallel decoding. In *CVPR*, 2021. 1, 2, 4, 5, 6, 7, 8

[49] Zhongdao Wang, Liang Zheng, Yixuan Liu, Yali Li, and Shengjin Wang. Towards real-time multi-object tracking. In *ECCV*, 2020. 3

[50] Chao-Yuan Wu and Philipp Krahenbuhl. Towards long-form video understanding. In *CVPR*, 2021. 2

[51] Chao-Yuan Wu, Christoph Feichtenhofer, Haoqi Fan, Kaiming He, Philipp Krahenbuhl, and Ross Girshick. Long-term feature banks for detailed video understanding. In *CVPR*, 2019. 2

[52] Chao-Yuan Wu, Yanghao Li, Karttikeya Mangalam, Haoqi Fan, Bo Xiong, Jitendra Malik, and Christoph Feichtenhofer. MeMVit: Memory-augmented multiscale vision transformer for efficient long-term video recognition. In *CVPR*, 2022. 2

[53] Yuhuai Wu, Markus N Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *ICLR*, 2022. 2

[54] Yilei Xiong, Bo Dai, and Dahua Lin. Move forward and tell: A progressive generator of video descriptions. In *ECCV*, 2018. 8

[55] Shen Yan, Xuehan Xiong, Anurag Arnab, Zhichao Lu, Mi Zhang, Chen Sun, and Cordelia Schmid. Multiview transformers for video recognition. In *CVPR*, 2022. 1

[56] Antoine Yang, Arsha Nagrani, Paul Hongsuck Seo, Antoine Miech, Jordi Pont-Tuset, Ivan Laptev, Josef Sivic, and Cordelia Schmid. Vid2seq: Large-scale pretraining of a visual language model for dense video captioning. In *CVPR*, 2023. 1, 2, 3, 4, 5, 6, 7, 8

[57] Jiahui Yu, Zirui Wang, Vijay Vasudevan, Legg Yeung, Mojtaba Seyedhosseini, and Yonghui Wu. Coca: Contrastive captioners are image-text foundation models. *TMLR*, 2022. 3

[58] Rowan Zellers, Jiasen Lu, Ximing Lu, Youngjae Yu, Yanpeng Zhao, Mohammadreza Salehi, Aditya Kusupati, Jack Hessel, Ali Farhadi, and Yejin Choi. Merlot reserve: Neural script knowledge through vision and language and sound. In *CVPR*, 2022. 6, 8

[59] Yue Zhao and Philipp Krähenbühl. Real-time online video detection with temporal smoothing transformers. In *ECCV*, 2022. 3, 6

[60] Yucheng Zhao, Chong Luo, Chuanxin Tang, Dongdong Chen, Noel Codella, and Zheng-Jun Zha. Streaming video model. In *CVPR*, 2023. 3

[61] Luowei Zhou, Chenliang Xu, and Jason Corso. Towards automatic learning of procedures from web instructional videos. In *AAAI*, 2018. 2, 5, 8

[62] Luowei Zhou, Yingbo Zhou, Jason J Corso, Richard Socher, and Caiming Xiong. End-to-end dense video captioning with masked transformer. In *CVPR*, 2018. 1, 2, 4, 7

[63] Wanrong Zhu, Bo Pang, Ashish V Thapliyal, William Yang Wang, and Radu Soricut. End-to-end dense video captioning as sequence generation. *ACL*, 2022. 2, 6, 7