Video Summarization Using AI

Review Report- 2

1st November 2024 - 2nt December 2024

Student:

Uchit N M - PES1UG22CS661

Faculty Mentors:

Prof. Prasad H B prasadhb@pes.edu

Dr. Gowri Srinivasa gsrinivasa@pes.edu

Prof. Preet Kanwal preetkanwal@pes.edu

Table of Contents

Introduction:

This report provides a detailed review of the research paper "VideoXum: Cross-modal Visual and Textural Summarization of Videos", and is organized into two main sections. Section 1 provides a **compression of the comprehensive literature**, summarizing key findings and methodologies. Section 2 presents a simple **code flow overview** of the implementation of the relevant paper.

Section 1 - Comprehensive literature Review:

VideoXum: Cross-modal Visual and Textural Summarization of Videos

Authors : Jingyang Lin, Hang Hua , Ming Chen , Yikang Li Jenhao Hsiao , Chiuman Ho , Jiebo Luo

Date of publish : 23 April 2024

Link : https://arxiv.org/pdf/2303.12060

About the paper :

The Authors introduce a novel cross-modal video summarization task, which involves generating visual and textual summaries with semantic coherence. To achieve this the authors propose VTSUM-BLIP, an end-to-end framework built on BLIP (Bootstrapped Language-Image Pretraining) to handle both modalities simultaneously, maintaining and improving semantic coherence between the two outputs.

Challenges in multimodal approach:

i. Benchmark Scarcity: The lack of large-scale, diverse, and well-annotated datasets hampers research and innovation in cross-modal video summarization.

ii. Optimization Challenges: Balancing and stabilizing training processes for multiple modalities is difficult, as optimizing one modality might detract from the other. Stable training enhances single-modal learning and overall performance.

iii. Semantic Coherence: Ensuring and evaluating the alignment between video and text summaries remains a complex task, as it requires maintaining meaningful correspondence across modalities.

Addressing the challenges :

The Benchmark Scarcity Challenge -

VideoXum is a large-scale dataset for **cross-modal video summarization**, built on the **ActivityNet Captions** benchmark, which includes **200 activity categories** across five top-level topics: Eating and Drinking, Sports, Socializing, Personal Care, and Household. It contains **14,001 videos** and **140,000 aligned video-text summary pairs**, enabling the transition from single-modal to cross-modal summarization. The dataset is curated from **20K real-life YouTube videos**, featuring diverse content and dense temporal annotations, with each segment paired with a sentence description for video understanding and generation.

The original video captions, covering **94.6% of video length**, were too long for concise summaries, so **reannotation** was performed to create **shortened summaries** (around 15% of the original length) aligned with the text. A **20% length compression ratio** was applied to filter videos.

The dataset's statistics include videos ranging from **10 to 755 seconds**, with an average length of **124.2 seconds**. Summaries are compressed to **13.6% of the original video length**, and text summaries average **49.9 words**. The dataset is divided into **training (8,000)**, **validation (2,001)**, and **test (4,000)** sets, for cross-modal tasks.

Comparison with Existing Single-modal Video Summarization Datasets and Dataset validation:

TABLE I : Comparison with existing single-modal video-to-video summarization and video-to-text summarization datasets.

Dataset	Domain	# Videos	Avg Ratio (%) VideoSum	Avg Len TextSum	Supported Task V2V-Sum	V2T-Sum	V2VT-Sum
MSVD [31]	open	1,970	-	8.7	✗	✓	✗
YouCook [6]	cooking	88	-	15.9	✗	✓	✗
MSR-VTT [7]	open	7,180	-	18.6	✗	✓	✗
ActivityNet [8]	open	20,000	-	40	✗	✓	✗
SumMe [4]	3 categories	25	15% <	-	✓	✗	✗
Youtube [5]	5 categories	50	15% <	-	✓	✗	✗
TVSum [3]	10 categories	50	15% <	-	✓	✗	✗
VideoXum (ours)	open	14,001	13.6%	49.9	✓	✓	✓

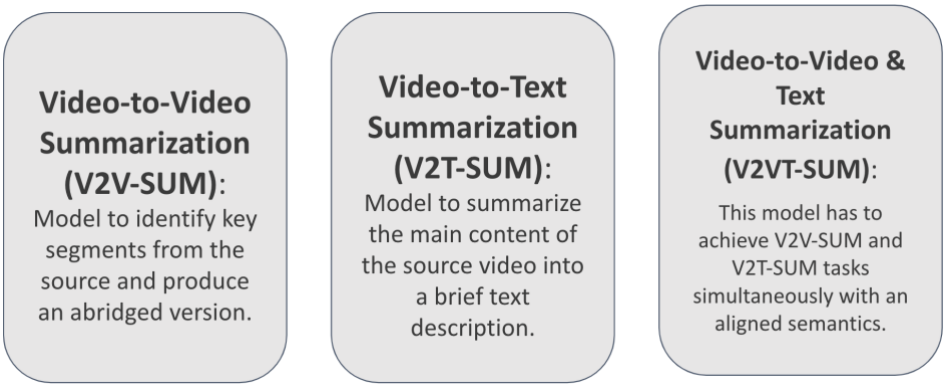
TABLE II : Comparison of F1 score on human annotations. F1-avg and F1-Max score across all reference summaries:

VideoXum (ours)		SumMe		TVSum	
F1-Avg	F1-Max	F1-Avg	F1-Max	F1-Avg	F1-Max
36.2	59.5	31♯	54♯	54♯	78♯

The **VideoXum** dataset stands out by offering **aligned video and text summaries**, unlike existing datasets that focus on single-modal summaries. It is significantly larger than benchmarks like **SumMe** and **TVSum**, providing more data and diverse, open-domain scenarios. The dataset ensures **high-quality annotations** through a **leave-one-out evaluation strategy**, with annotation quality found to be comparable to existing datasets, as shown in **TABLE II**. Thus making VideoXum a suitable resource for cross-modal video summarization tasks.

The Optimization Challenges and Semantic Coherence Challenge -

To address this, A **V2X-SUM** (**Fig 1**) models are designed and proposed, here X refers to **V- Video**; **T-Text** and **VT- Video&Text** which defines a multi cross modal approaches is proposed in which video and text summarization is given parallelly using the same feature vectors form a shared encoder.



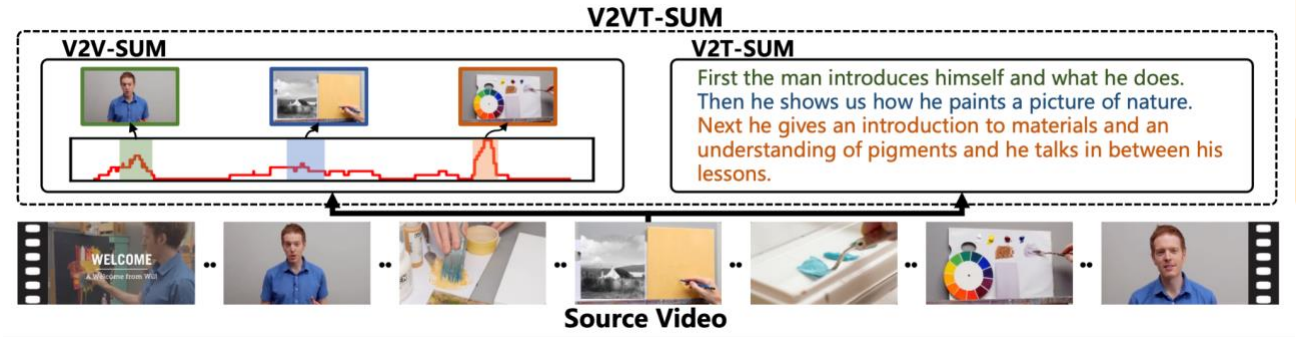


Fig 1. Illustration of the V2X-SUM tasks

The Baseline approach:

The VTSUM-BLIP framework is adopted as an end-to-end, cross-modal, modular approach.

Why BLIP? The BLIP framework in the VideoXum dataset due to its powerful ability to handle both **vision** and **language** tasks, which **generate semantically aligned summaries**, and its **pretraining advantage make it attractive for the complex task of cross-modal video summarization.**

This VLP encoder-decoder architecture of the BLIP provides superior initialization aids in designing an efficient hierarchical video encoding strategy, incorporating a frozen encoder, a temporal modeling module, and a context aggregation module to encode long videos. The video encoder is followed by different task-specific decoders for video and text summarization.

This approach gives advantages in terms of flexibility to perform more complex downstream without changing the structure of the pretrained backbone. And also enables joint training of the video and text summarization decoders in parallel.

In terms of evaluation of metrics, An evaluation metric termed VT-CLIPScore on the VideoXum benchmark to evaluate cross-modal semantic consistency. The empirical results show the high consistency of our proposed metric with human evaluation. Refer Section -

The Methodology :

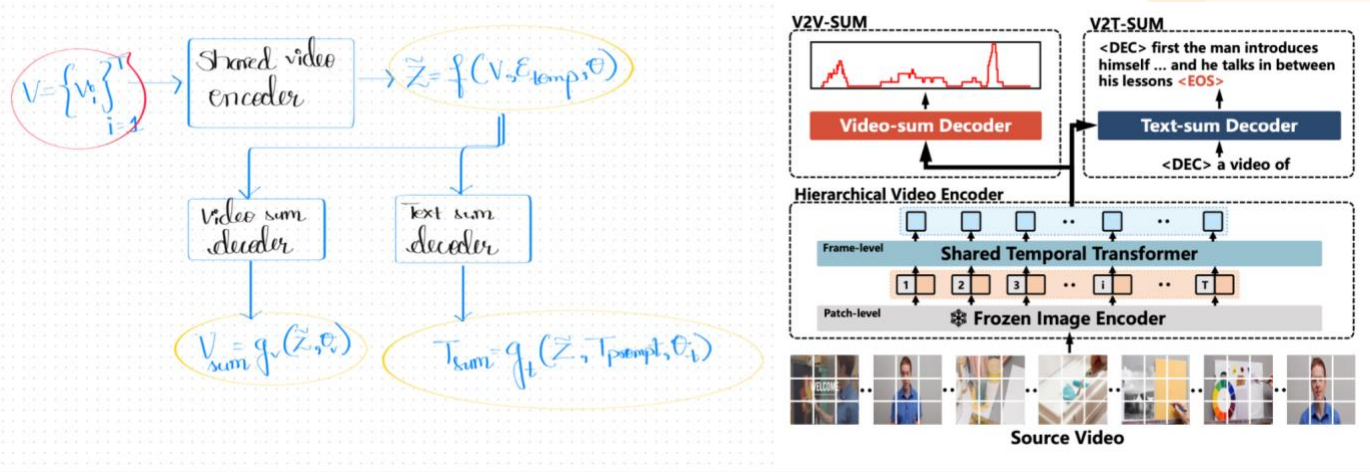


Fig 2. Problem Formulation (left) and Propped flow of the architecture(Right)

Consider the problem formulation diagram from Fig.2 (left); Here given a video $V= \{v_i\}^T_{i=1}$, T is the number of frames in the video and v_i denotes the i-th frame in the temporal order. The object is to generate shared video encoder $f(\cdot; \theta)$ followed by two task-specific decoders, including a video summarization (video-sum) decoder $g_v(\cdot; \theta_v)$ and a text summarization (text-sum) decoder $g_t(\cdot; \theta_t)$. where θ, θ_v and θ_t represent the learnable parameters of the shared video encoder, and corresponding decoders, respectively.

$$\tilde{\mathcal{Z}} = f(\mathcal{V}, \mathcal{E}_{\text{temp}}; \theta),$$

The above depicts the **output ($\tilde{\mathcal{Z}}$)** i.e spatial temporal feature encoding of the input video from the shared encoder. Where ε_{temp} is temporal position embedding for the corresponding video frames.

Then,

$$\mathcal{V}_{\text{sum}} = g_v(\tilde{\mathcal{Z}}; \theta_v),$$
$$\mathcal{T}_{\text{sum}} = g_t(\tilde{\mathcal{Z}}, \mathcal{T}_{\text{prompt}}; \theta_t),$$

Represent the video summarization and text summarization respectively.

Here, T_{Prompt} denotes a prompt sequence.

The completer Architecture :

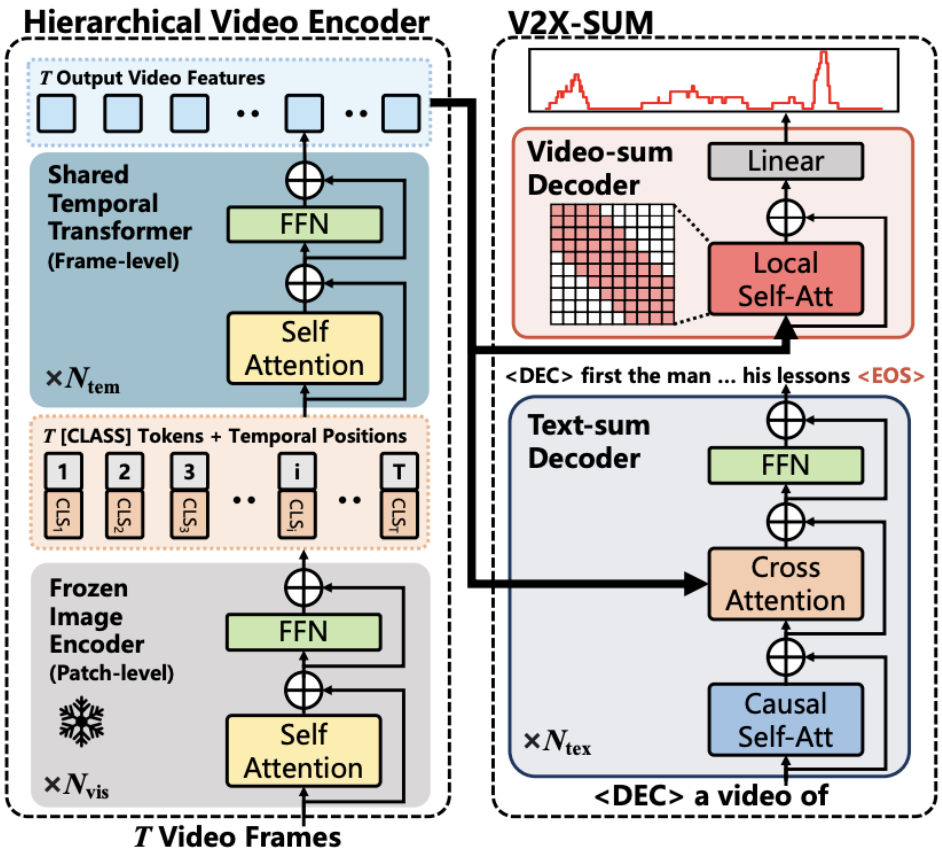
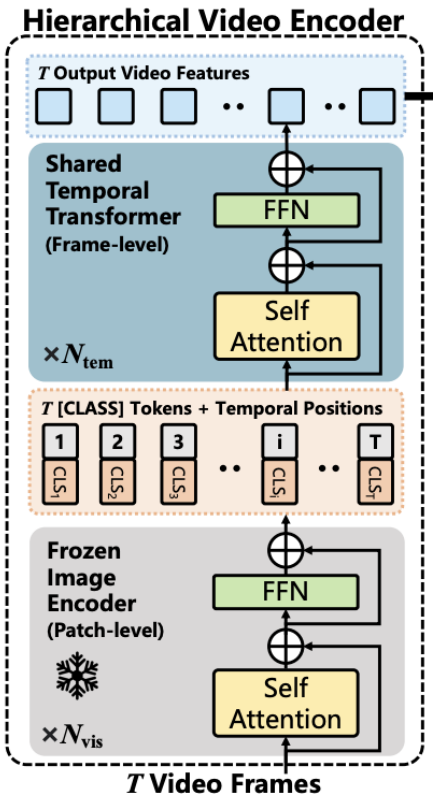


Fig 3. VTSUM-BLIP

The Hierarchical encoder structure of the shared encoder:



$$\tilde{\mathcal{Z}} = f(\mathcal{V}, \mathcal{E}_{\text{temp}}; \theta),$$

Fig 4. Hierarchical encoder with Frozen Image encode and Shared TT structure.

The hierarchical video encoder f(·;θ) is designed to efficiently extract spatiotemporal visual features from long videos. It adapts the BLIP image encoder into a hierarchical structure, enabling the handling of long video sequences without altering the core encoder architecture. The process starts with a frozen image encoder, which projects each video frame v_i into a shared representation space as patches of defined sizes, producing a sequence of visual tokens $\tilde{Z} = \{zi\}^{T}_{i=1}$ where T is the number of frames. Then, temporal position embeddings $\epsilon_{temp} = \{e\ i\}^{T}_{i=1}$ are added to the visual tokens, which are passed through a shared Temporal Transformer (TT) to model the temporal relationships between frames. The output

is a sequence of spatiotemporal visual features ($\tilde{Z}$), combining both spatial and temporal information from the video.

Video-Sum Decoder :

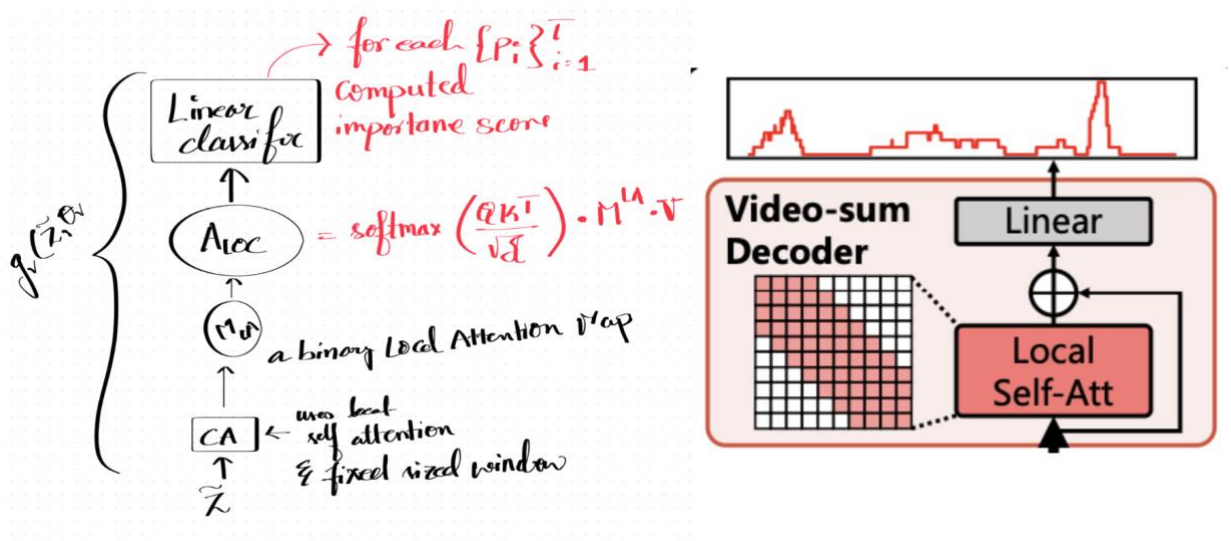


Fig 5. Video-Sum Decoder

The **video-sum decoder** $g_v(\cdot;\theta v)$ incorporates a **Context Aggregation (CA) module**. This module leverages **local self-attention** to capture contextual information from neighboring frames. A **fixed-size slice window** is defined for each temporal position, constructing a **binary local attention map** $M_{LA} \in \{0,1\}_{T\times T}$ with a specified window size ε. (Refer Fig. 6 for M_{LA})

The **MLA matrix** is a binary T×T matrix where T is the total number of video frames. Frames within the slice window (i−ε/2 to i+ε/2) are set to **1** (attended), while those outside are set to **0** (ignored). This results in a sparse matrix with ones in the region of interest and zeros elsewhere.

eg. Consider Total number of frames (T): 10 and Window size (ϵ): 3

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Fig 6. M_{LA} - Binary Local Attention Map

Finally, we select the top 15% of frames to attain a video-sum result V_{sum}.

Text-Sum Decoder:

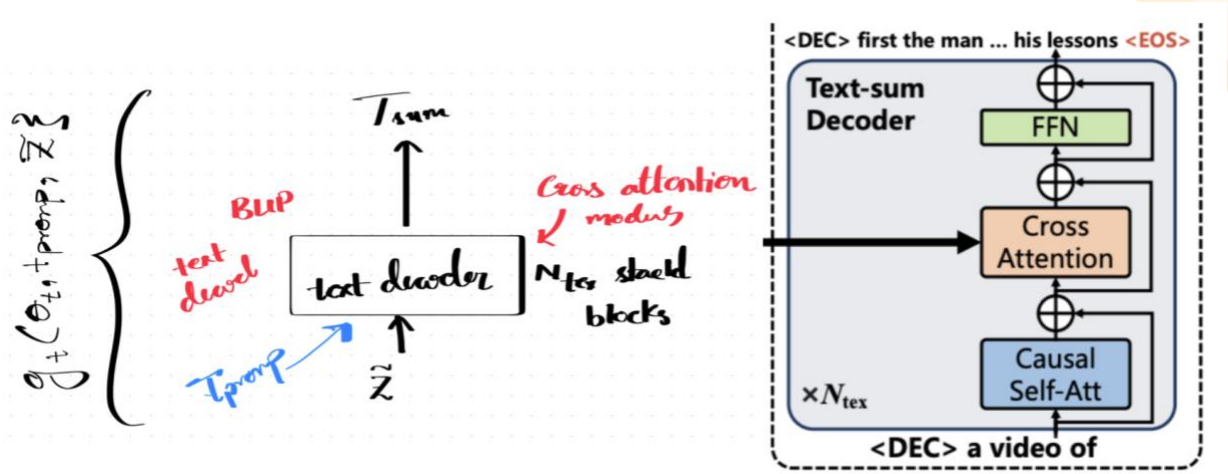
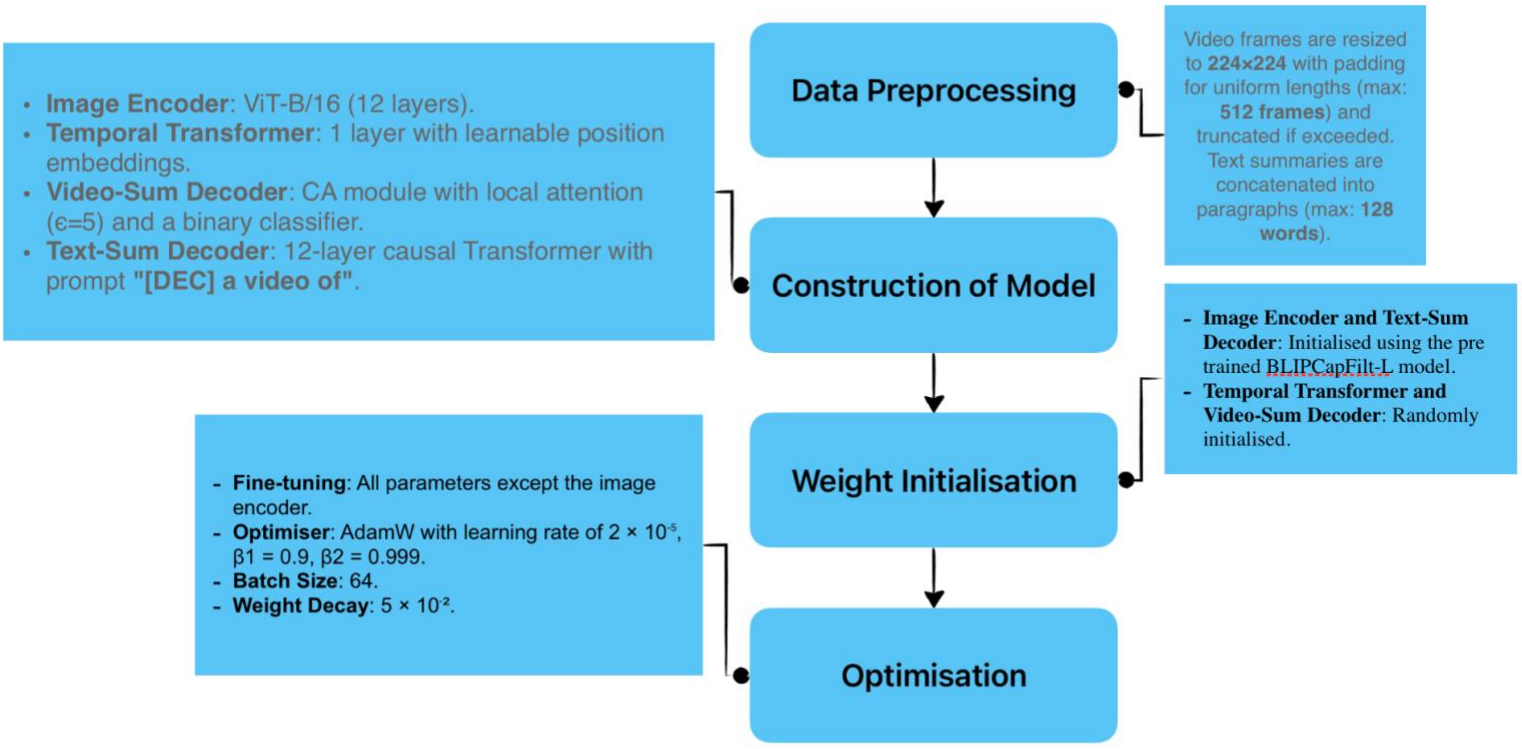


Fig 7. Text-Sum Decoder

The pretrained **BLIP text decoder** serves as a strong baseline for text generation. The **text summarization decoder** $g_t(\cdot;\theta_t)$ comprises N_{tex} stacked Transformer decoder blocks with cross-attention modules. During decoding, the text decoder uses a prompt sequence T_{Prompt} and video features from the video encoder as inputs to generate the final text summary T_{sum}.

Experimental Setup and Implementation Details:



Evaluation:

Video Summary Evaluation:

- **F1 Score**: Measures the balance between precision and recall in the video summary.
- **Kendall's τ and Spearman's ρ**: Rank correlation metrics to evaluate how well predicted frame rankings match true rankings.

Text Summary Evaluation:

- **BLEU**: Evaluates n-gram precision between generated and reference text.
- **METEOR**: Considers synonyms, stemming, and word order for more nuanced text evaluation.

- **ROUGE-L**: Focuses on recall by measuring the longest common subsequence (LCS).
- **CIDEr**: Assesses term frequency consensus across multiple references.

Semantic Consistency Evaluation:

- **VT-CLIPScore**: Adapts and fine-tunes the CLIP model on the VideoXum dataset using contrastive learning to measure semantic alignment between video and text summaries.
- **CLIPScore**: Evaluates video-text alignment, fine-tuned for domain-specific reliability.

The CLIP model was fine-tuned with AdamW for **50 epochs** using a batch size of 16 for validating the score.

Table III infers the Human evaluation of the baseline models on the VideoXum dataset and TABLE IV to the outcomes of cross-modality similarity under different semantic changes thus, VT-CLIPScore demonstrates improved sensitivity by better distinguishing **positive pairs** (aligned video-text summaries) from **negative pairs** (unpaired video-text summaries).

Method	V2V-SUM	V2T-SUM		Cross-Modal
	Accuracy	Accuracy	Fluency	Consistency
VTSUM-BLIP (Base)	3.1	4.1	4.3	3.2
+ Temporal Transformer	3.5	4.2	4.2	3.2
+ Context Aggregation	3.2	4.1	4.3	3.1
+ TT + CA	**3.8**	**4.4**	**4.4**	**3.4**

TABEL III - Human evaluation of the baseline models on the VideoXum dataset

Method	Cross-Modal Similarity (Cosine Similarity)	
	Positive pairs	Negative pairs
CLIPScore [17]	14.5	0.3
VT-CLIPScore	38.0	0.2

TABLE IV - Results of cross-modality similarity under different semantic changes.

Experimental Results and Analysis

The VTSUM-BLIP framework, with Temporal Transformer (TT) and Context Aggregation (CA), gives substantial results across V2V-SUM, V2T-SUM, and VTSUM tasks. The end-to-end approach proved advantage over the two-stage method, minimizing error propagation.

On external benchmarks (TVSum, SumMe, ActivityNet Captions), BLIP-VSUM performs competitively, with TT and CA improving video summarization. In text summarization, the model gives results over baselines in BLEU, METEOR, ROUGE-L, and CIDEr. (TABLE V and TABLE VI)

Human evaluation on 50 samples shows VTSUM-BLIP generates more fluent summaries, with a consistency of 0.49. While human annotators perform better in most metrics, VTSUM-BLIP is competitive for V2T-SUM.

VT-CLIPScore outperforms CLIPScore, offering improved sensitivity and reliability in measuring cross-modal semantic consistency, as shown by higher cosine similarity for positive pairs.

Method	V2V-SUM			V2T-SUM				V2VT-SUM
	F1 score	Kendall	Spearman	BLEU@4	METEOR	ROUGE-L	CIDEr	VT-CLIPScore
Frozen-BLIP	16.1	0.008	0.011	0.0	0.4	1.4	0.0	19.5
Single-Modal Video Summarization								
Video-to-Video Summarization								
VSUM-BLIP (Base)	21.7	0.131	0.207	-	-	-	-	-
+ Temporal Transformer	22.1	0.168	0.222	-	-	-	-	-
+ Context Aggregation	22.2	0.172	0.228	-	-	-	-	-
+ TT + CA	23.1	0.185	0.246	-	-	-	-	-
Video-to-Text Summarization								
TSUM-BLIP (Base)	-	-	-	5.5	11.7	24.9	18.6	-
+ Temporal Transformer	-	-	-	5.6	11.8	24.9	20.9	-
Cross-Modal Video Summarization								
Two-stage Manner	19.4	0.107	0.143	5.1	11.2	24.0	15.6	28.2
VTSUM-BLIP (Base)	21.7	0.131	0.207	5.5	11.7	24.9	18.6	28.4
+ Temporal Transformer	22.4	0.176	0.233	5.7	12.0	24.9	22.4	28.9
+ Context Aggregation	22.2	0.172	0.228	5.5	11.7	24.9	18.6	28.6
+ TT + CA	**23.5**	**0.196**	**0.258**	**5.8**	**12.2**	**25.1**	**23.1**	**29.4**
Human	33.8	0.305	0.336	5.2	14.7	25.7	24.2	38.0

TABLE V: Performance of the baseline models on the VideoXum dataset test set for three different: V2XSum tasks

Method	ActivityNet Captions			
	BLEU@4	METEOR	ROUGE-L	CIDEr
DENSE [8]	1.6	8.9	-	-
DVC-D-A [84]	1.7	9.3	-	-
Wang *et al.* [34]	2.3	9.6	19.1	12.7
Bi-LSTM+TempoAttn [35]	2.1	10.0	-	-
Masked Transformer [35]	2.8	11.1	-	-
Support-Set [70]	1.5	6.9	17.8	3.2
TSUM-BLIP (Base)	5.5	12.1	25.1	19.7
+ Temporal Transformer	**5.7**	**12.1**	**25.2**	**22.2**

Table VI: Comparison with state-of-the-art methods on the ActivityNet Captions dataset

Extended Discussions:

The **VTSUM-BLIP** model has varying computational complexities depending on the task. The model sizes are 435.6 MB for V2V-SUM, 564.8 MB for V2T-SUM, and 567.1 MB for V2VT-SUM. For the **V2V-SUM** task, the local window size ε=5 provided the best performance, balancing context without introducing irrelevant frames. The number of **Temporal Transformer layers** did not significantly affect performance, so only one layer was used. Finally, while human performance outperforms the model on **V2V-SUM** and **V2VT-SUM**, the model performs similarly to humans on **V2T-SUM**, particularly in **BLEU score**, showing strong results in text summarization.

Section 2- Code Overview:

Cross-modal approach:

This script implements cross-modal visual and textual summarization using temporal transformers and vision transformers to handle video data. The workflow integrates dataset processing, model creation, training, and evaluation for summarization tasks. (Fig 8a - Fig 8f represent the flow)

Main Function - train_v2vt_sum.py:

The program initializes by parsing command-line arguments and preloading parameters such as temporal transformer (TT) depth, weight files, and batch size from configuration files. If distributed processing is enabled, a device distributor is set up, and a seed value is initialized for reproducibility. The main function ensures compatibility with distributed systems and sets default values when distributed sampling is not used.

Dataset Creation

The dataset is divided into three subsets: train, val, and test, created using the create_dataset method. Each dataset undergoes normalization, followed by transformations like RandomResizeCrop, RandomHorizontalFlip, and RandomAugment, and is converted into tensors.

If the dataset type is VideoXum, the ActivityNet class is used, which processes video annotations, embeds clip features, and organizes data in directories. It includes utility methods to handle video embeddings, masks, and labels. For distributed systems, PyTorch's DistributedSampler ensures proper data partitioning. The createLoader method generates data loaders with specified batch sizes, worker counts, and a collate_function to batch samples efficiently.

Model Creation

Two main model types are implemented:

1. vtsum_blip_tt: Focuses on visual and temporal transformers for video and text summarization.
2. vtsum_blip_tt_ca: Extends vtsum_blip_tt by adding context aggregation (CA) layers.

The base class, VTSUM_BLIP_TT, includes visual transformers, positional embeddings, temporal transformers, and decoders for summarization. The temporal transformer is created using create_tt, with configurable parameters like width (768 for base, 1024 for large), depth, drop path rate, and checkpointing. The decoder initializes a video summarization head (vsum_head) with specific loss functions and configurations.

A BERT model aids textual summarization, with sampling techniques like Nucleus Sampling (top-p sampling) or Beam Search, based on configuration. Nucleus sampling selects tokens based on cumulative probabilities (top_p=0.9), while beam search retains the most probable sequences (num_beams=3).

The VideoCLIP class, built on CLIP, processes video frames using a vision transformer (ViT-B/16). It handles frame tokenization, text encoding, and attention mechanisms. The forward method processes video frames through layers while managing positive and negative frame pairs.

Finally, the AdamW optimizer is employed with defined learning rates and weight decay for effective model training.

Training

During training, metrics like learning rate, loss, and aggregated performance are tracked using a MetricLogger. The process iterates over the dataset, fetching video embeddings, masks, and labels for video (vsum_labels) and text summarization (tsum_labels). The model computes losses (loss_tsum and loss_vsum), which are aggregated before clearing and updating gradients. Training runs for multiple epochs, saving checkpoints at defined intervals. If distributed systems are enabled, processes are synchronized.

Evaluation

The evaluation phase begins with metric computation using Kendall and Spearman rank correlation coefficients. Kendall and Spearman rank the relationship between ranked variables (-1 to 1).

Saliency scores are calculated by selecting the top 15% of video frames based on sorted scores. Average top-frame features are normalized and scored using dot products to compute similarity

between video and text features. CLIP-based scoring further evaluates alignment between video and text embeddings.

The COCO evaluation module assesses summarization quality using metrics like BLEU, METEOR, ROUGE_L, and CIDEr. The results include precision, recall, and alignment scores, providing a comprehensive evaluation of video-text summarization

The combination of these metrics, saliency-based selection, and CLIP scoring gives a basis for evaluation of the cross-modal summarization.

Flow Diagram :

The main flow:

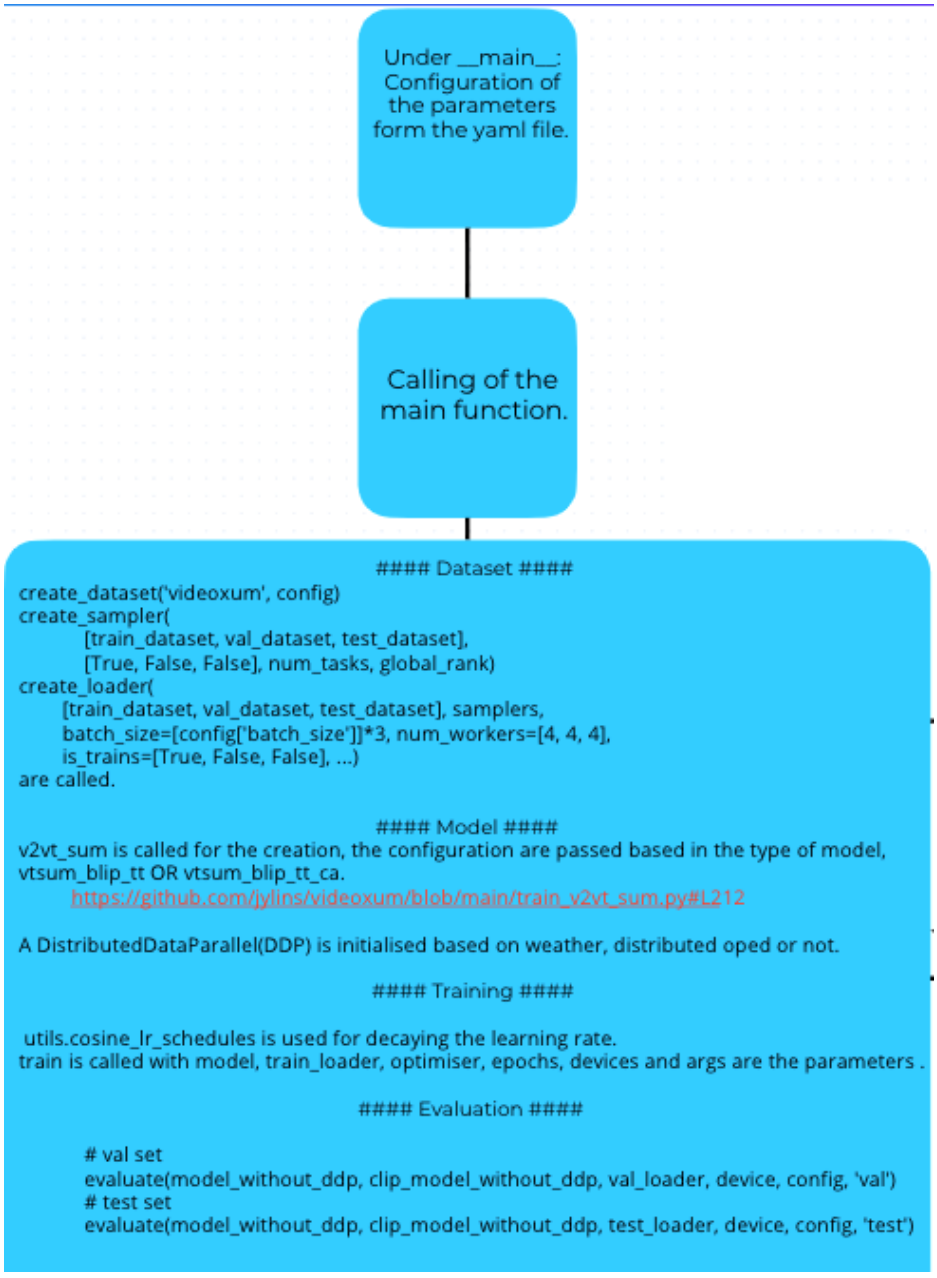


Fig 8.a main function of train_v2vt_sum.py

The script train_v2vt_sum.py is designed for training video-text summarization models. It incorporates multiple processes, including dataset creation, model initialization, training, and evaluation, utilizing PyTorch frameworks and distributed computing when available.

Dataset Creation :

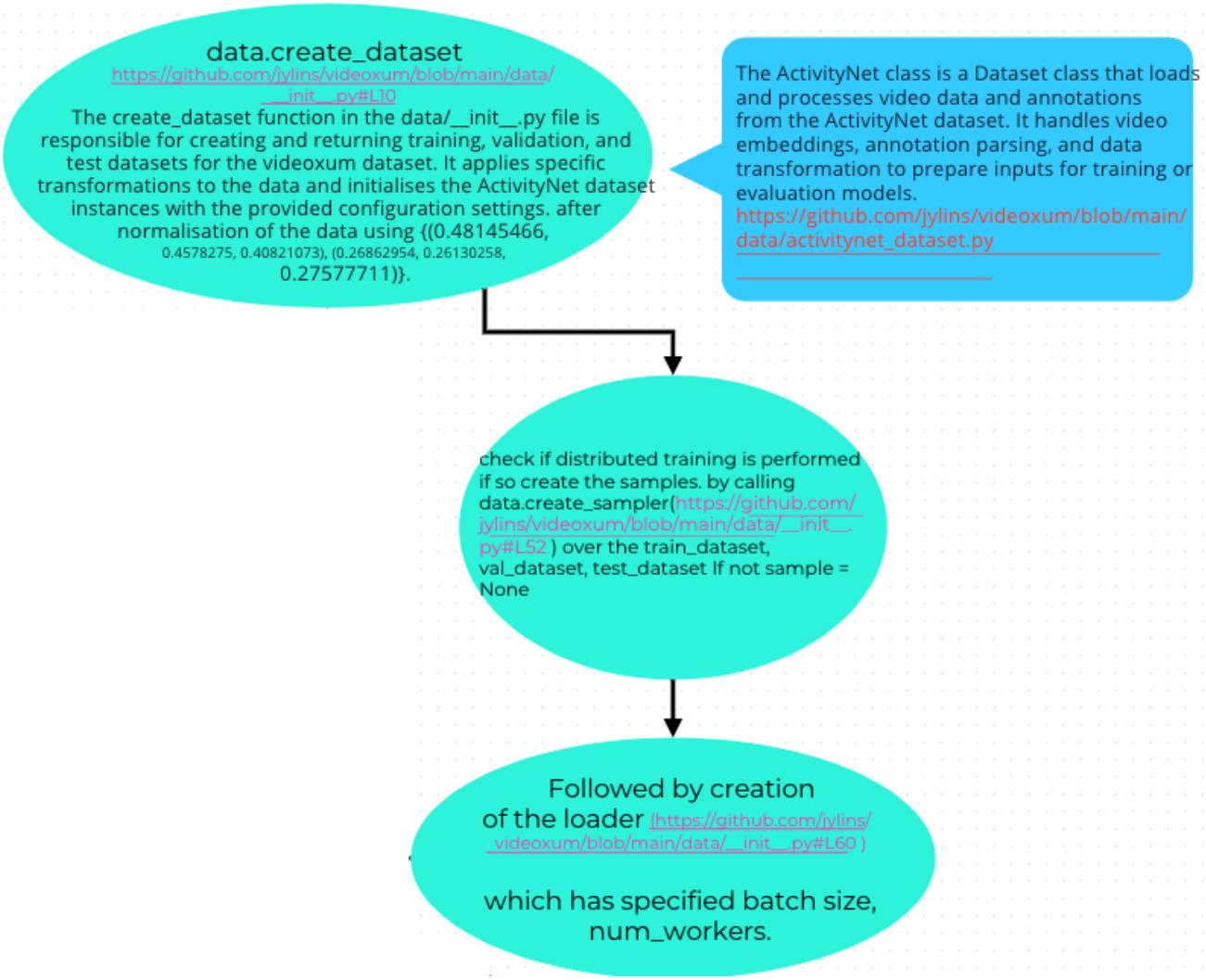


Fig 8.b - Dataset flow

Model Creation:

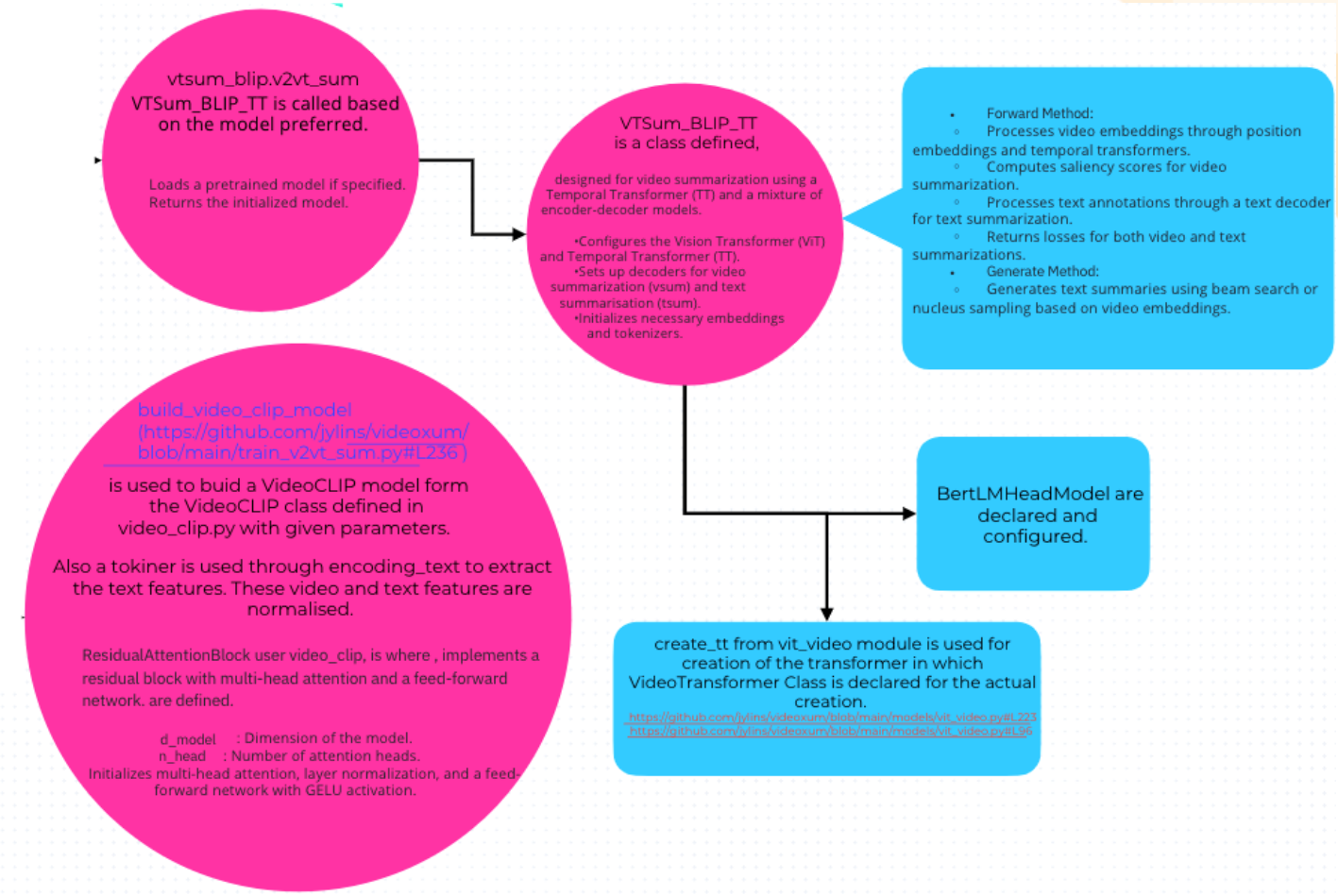


Fig 8.c - Model flow

Training:

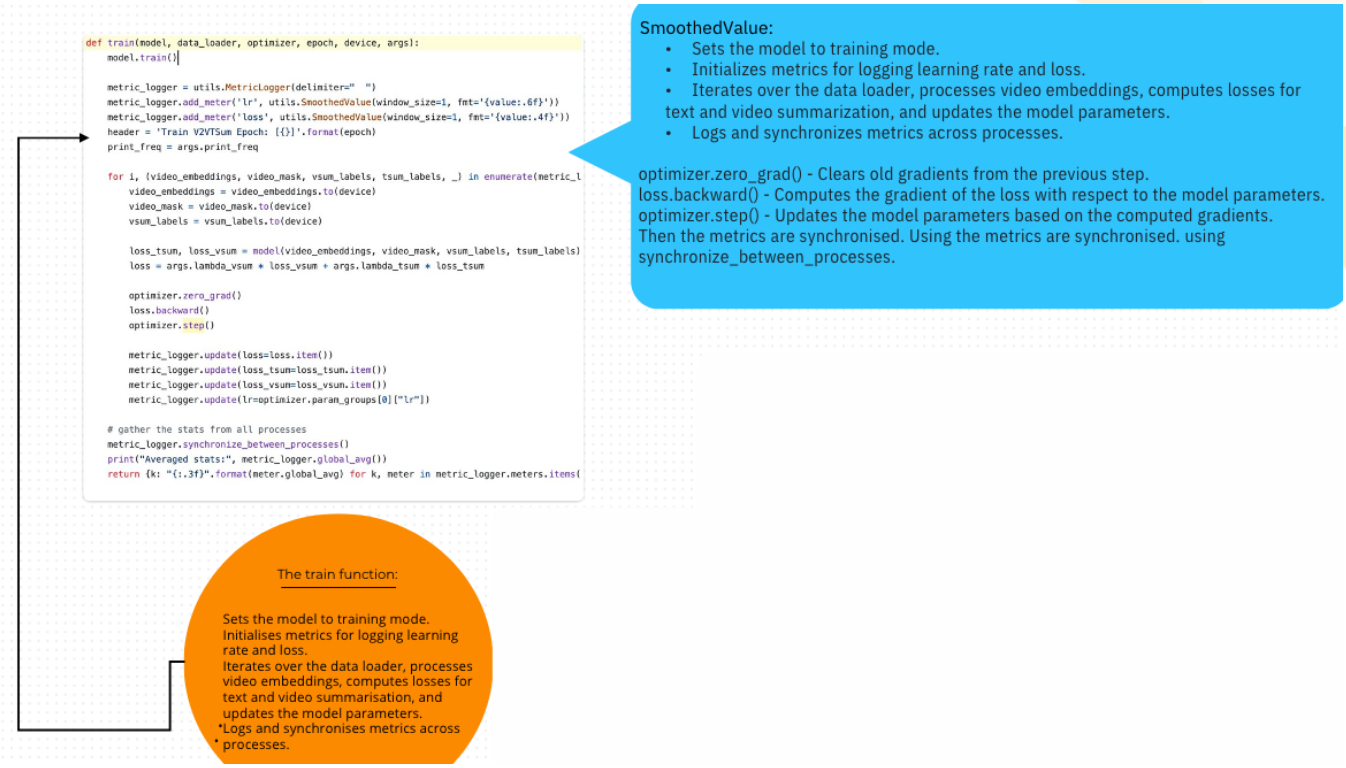


Fig 8.d - Training flow

Evaluation:

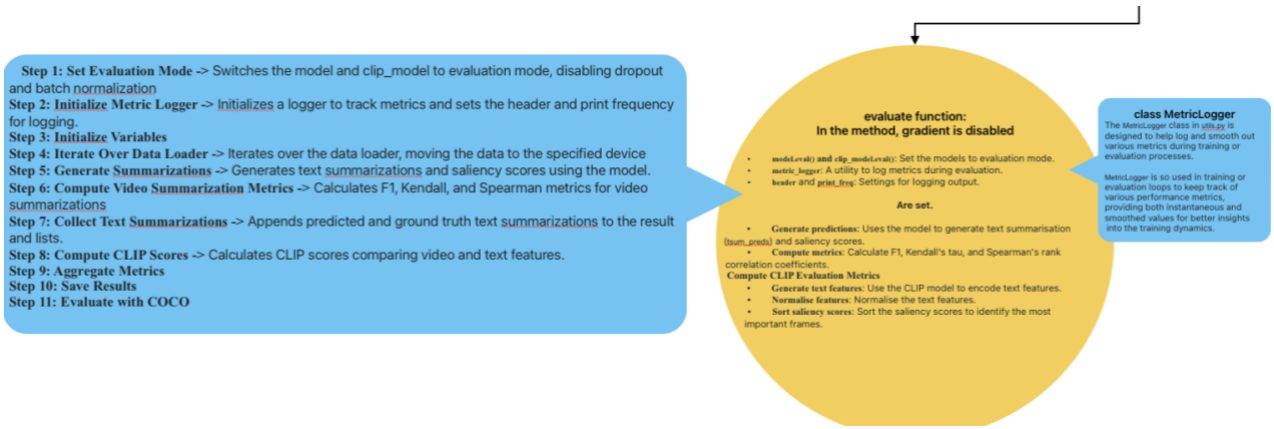


Fig 8.e - Evaluation flow

Fig 8 : Complete Chart can be accessed here :
https://drive.google.com/file/d/1A8I4nznZiffNI6hBXm2NXHUsB8GpopXi/view?usp=sharing

References and Links:

[1]J. Lin *et al.*, "VideoXum: Cross-modal Visual and Textural Summarization of Videos," *arXiv.org*. https://arxiv.org/abs/2303.12060

VideoXum: Cross-modal Visual and Textural Summarization of Videos - https://arxiv.org/pdf/2303.12060

[2]jylins, "GitHub - jylins/videoxum: [TMM 2023] VideoXum: Cross-modal Visual and Textural Summarization of Videos," *GitHub*. https://github.com/jylins/videoxum .

Github Source code - https://github.com/jylins/videoxum

Video Summarization Using AI

Review Report- 2

1st November 2024 - 2nd December 2024

Students:

Arya Ajay Gupta - PES1UG22CS111

Faculty Mentors:

Prof. Prasad H B prasadhb@pes.edu

Dr. Gowri Srinivasa gsrinivasa@pes.edu

Prof. Preet Kanwal preetkanwal@pes.edu

Table of Contents

Introduction:

This report provides a detailed review of the research paper, “VideoSAGE: Video Summarization with Graph Representation Learning".

Comprehensive Literature Review:

VideoSAGE: Video Summarization with Graph Representation Learning

Authors : Jose M. Rojas Chaves, Subarna Tripathi

Date of publish : 14 April 2024

Link : https://arxiv.org/abs/2404.10539

About the paper :

The authors propose a graph-based representation learning framework for video summarization. First, they convert an input video into a graph where nodes correspond to individual video frames. They impose sparsity on the graph by connecting only pairs of nodes within a specified temporal distance. The video summarization task is then formulated as a binary node classification problem, where video frames are classified as either belonging to the output summary video or not. This graph construction captures long-range interactions among video frames while the sparsity ensures that the model avoids memory and compute bottlenecks during training. Experiments on two datasets (SumMe and TVSum) demonstrate the effectiveness of the proposed nimble model, which outperforms existing state-of-the-art summarization approaches while being an order of magnitude more efficient in terms of compute time and memory.

Goal of the Paper :

To tackle the challenges in video summarization, this research introduces a novel framework leveraging Graph Neural Networks (GNNs). The objective is to develop a model that:

- **Transforms videos into graph representations** where frames are treated as nodes, and edges encode temporal or contextual relationships.
- **Learn to distinguish informative frames** (nodes) from non-informative ones using binary node classification.
- **Outputs concise video summaries** that capture the most salient content, outperforming industry-standard models in efficiency and accuracy.

Graph Neural Networks (GNNs):

GNNs are neural networks specially designed to work with graphs—a type of data structure that models relationships between entities.

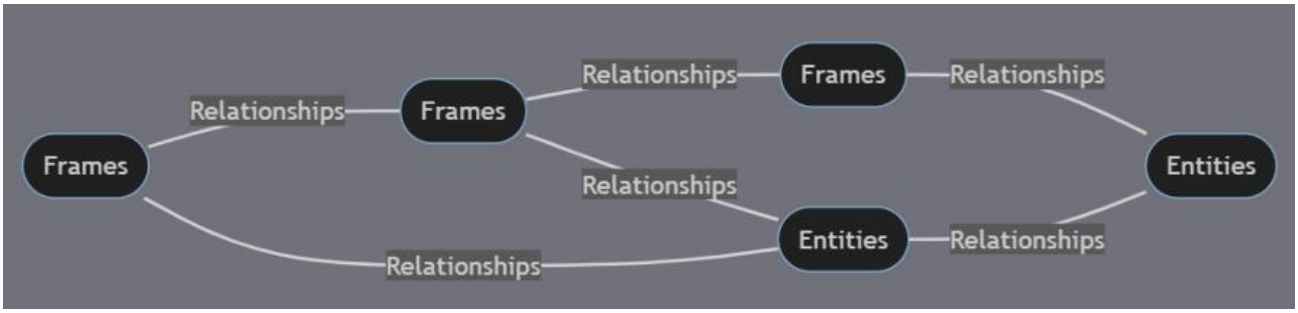


Fig 1 : Graphs representation

What is a Graph?

Graphs are mathematical structures used to model relationships between objects. They are widely used in various domains, from social networks to biology, and are fundamental in graph theory and computer science.

- A Graph is made up of:
  - **Nodes**: These represent the individual entities (e.g., video frames, people in a network).
  - **Edges**: These show the relationships or connections between nodes (e.g., temporal similarity, friendships)

Why Use GNNs?

- Traditional neural networks cannot handle the irregular structure of graph data.
- GNNs **capture patterns and relationships** between nodes, making them ideal for tasks like video summarization, social network analysis, and molecule modeling.

How Do GNNs Work? :

- **Message Passing:**
  - Nodes exchange information with their connected neighbors (e.g., frames share context with nearby frames).
  - This helps each node learn about its local and global relationships.
- **Feature Aggregation:**
  - Nodes combine information from neighbors to update their own features.
  - Repeated over multiple layers to capture deeper relationships in the graph.
- **Output:**
  - GNN outputs can represent the entire graph or specific nodes (e.g., key video frames).

Why GNNs Are Effective:

- **Context-Aware:** They consider relationships between frames.
- **Scalable**: Handle large graphs by focusing on meaningful connections.

Message Passing - How It Works :

Message passing in GNNs involves three key steps that repeat for several layers, enabling nodes to gather information from their neighbors over multiple hops:

1.**Message Computation:**

- Each node generates a message for its neighbors based on its own features.

2.**Message Passing:**

- Nodes exchange messages along edges (relationships).

3.**Message Aggregation at the Target Node:**

- Each node collects all incoming messages from its neighbors.

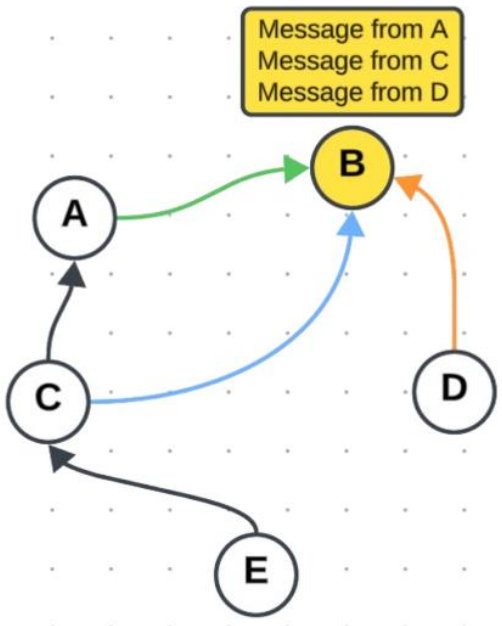


Fig 2 : Message Passing : In message passing, nodes compute and exchange information along edges, helping them learn from their neighbors

Feature Aggregation - How It Works :

Feature aggregation is a key step in updating a node's representation by incorporating information from its neighbors. Here's the process:

1. **Aggregating Incoming Messages**
    - Each node gathers messages from its neighboring nodes.
    - These messages are aggregated using one of the following methods:
        - **Mean:** Computes the average of all incoming messages.
        - **Sum:** Adds together all incoming messages.
        - **Max:** Selects the maximum value for each feature from the incoming messages.
2. **Combining with Current Features**
    - The aggregated information is integrated with the node's current features to update its representation.
3. **Propagation Across Layers**
    - The updated features are passed to the next layer in the Graph Neural Network (GNN).
    - This step is repeated across multiple layers, enabling each node to capture progressively more global relationships in the graph.

By iteratively aggregating and refining features, nodes can effectively learn representations that incorporate both local and global context over time.

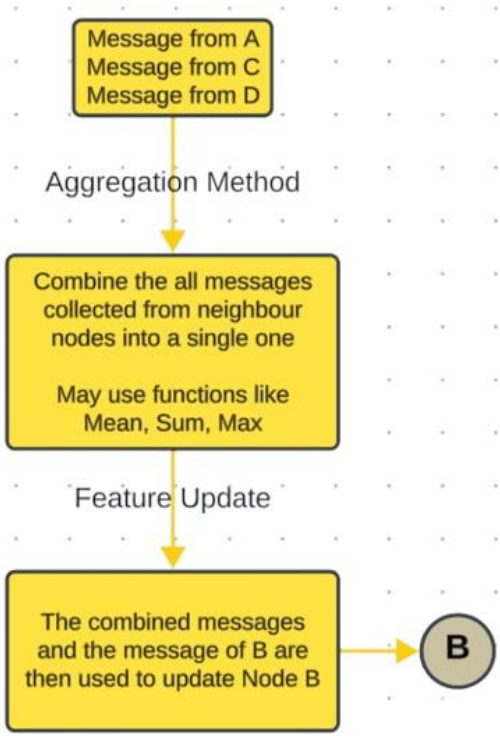


Fig 3 : Feature Update : Feature aggregation combines neighbor messages and updates node features for deeper relational understanding

Datasets:

- **SumMe Dataset:**
  - Composed of 25 raw videos ranging from 1 to 6.5 minutes.
  - Covers diverse themes like holidays, events, and sports.
- **TVSum Dataset:**
  - Includes 50 YouTube videos of 2 to 10 minutes each.
  - Spans 10 categories, sourced from the TRECVid Multimedia Event Detection dataset, including themes like sports, news, and DIY tutorials.

Key Features of the Datasets:

- Videos were sampled at 2 frames per second (fps) for processing.
- Feature representation extracted using GoogleNet's Pool5 layer (Size D=1024)
- Annotated by 15–20 human evaluators to score the importance of frames, later used to generate summaries.

Key Statistics:

- Total videos: 75 (SumMe + TVSum).
- Duration: 1 to 10 minutes per video.
- Resolution: Standard quality, suitable for analysis.

The Methodology :

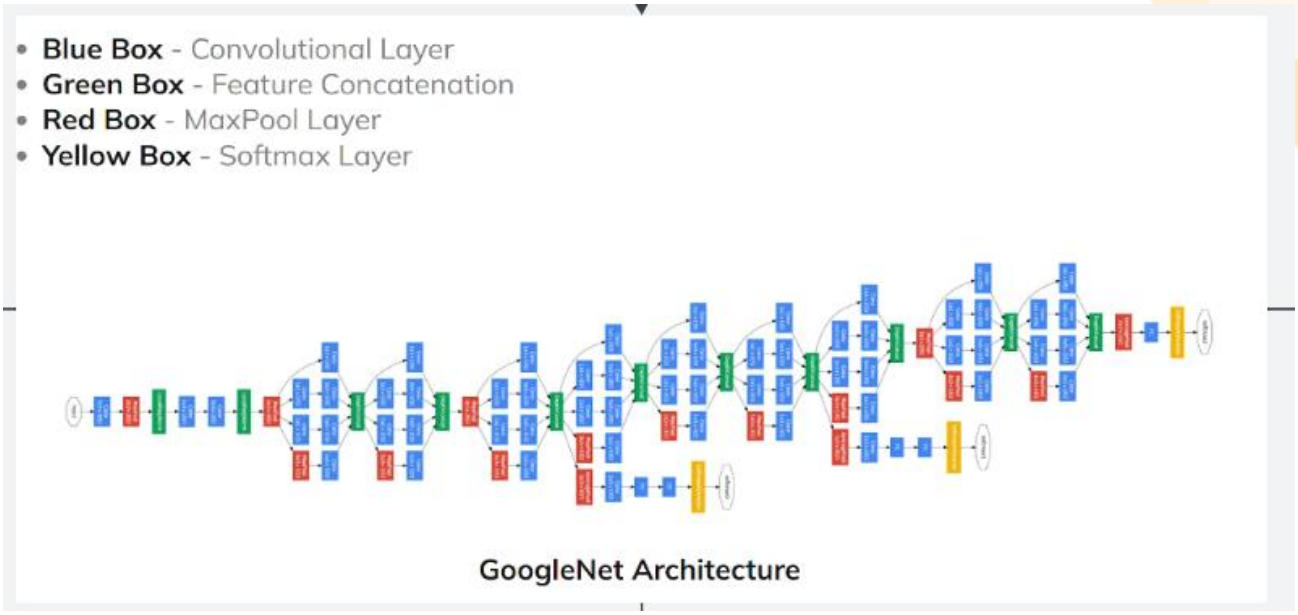


Fig 4. Problem Formulation (left) and Propped flow of the architecture(Right)

Preprocessing for GNN Input

1. **Frame Extraction**
   - Since GNNs cannot process video frames directly, the video is first submitted to a frame extraction pipeline, such as OpenCV or other open-source tools in Python.
   - The extracted frames are stored for further processing.
2. **Frame Resizing**
   - Each frame is resized to a resolution of 224 × 224 pixels, which is the ideal input size for the GoogleNet architecture.
3. **Feature Extraction Using GoogleNet**
   - GoogleNet, consisting of 22 parameterized layers and 5 non-parameterized layers, processes the resized frames.
   - The output of GoogleNet is a feature vector for each frame, capturing its visual and contextual information.
4. **Graph Construction**
   - In the constructed graph, each video frame corresponds to a node.
   - The information stored in each node is the feature vector obtained from GoogleNet, which serves as the input for the GNN.

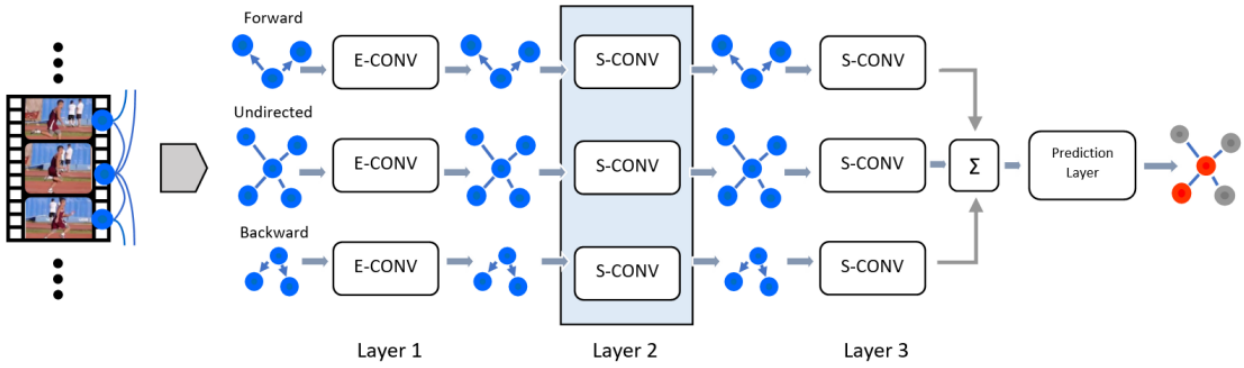


Fig 5 : Bi-Directional GNN model for Video Summarization

An illustration of utilized Bi-directional (a.k.a. Bi-dir) GNN model for video summarization. Here, we have three separate GNN modules for the forward, backward, and undirected graph, respectively. Each module has three layers where the weight of the second layer is shared by all three graph modules. The second layer is placed inside a solid-lined box to indicate the weight sharing while for the first and the third layer we use dotted-lines. E-CONV and S-CONV are shorthand for EDGECONV and SAGE-CONV, respectively.

Graph Construction for Video Summarization:

1. **Node Information**
   - Each video frame is represented as a node NNN in the graph.
   - The node information is the feature vector obtained from GoogleNet, encapsulating the frame's characteristics.
2. **Graph Types**

   Three types of graphs are constructed to capture temporal relationships among frames:
   - **Forward Graph**
     - Nodes are connected if the time difference between the two reference nodes equals T.
   - **Undirected Graph**
     - Nodes are connected if the time difference between frames lies within the range [−T,+T].
   - **Backward Graph**
     - Nodes are connected if the time difference between frames equals −T.

(Note: The paper does not specify the method to calculate the metric T, leaving it as a parameter to be determined.)

3. **Graph-Based Representation Learning**
    - **Lightweight GNN Layers**
        - The constructed graphs are submitted to a 3-layer lightweight GNN model.
        - The second layer employs shared weights across the three modules handling the forward, undirected, and backward graphs, ensuring temporal coherence.
    - **Aggregation Function**
        - Each GNN layer incorporates an aggregation function that updates the feature vectors of nodes based on their neighbors, refining the node representations through the graph structure.
4. This multi-graph approach captures diverse temporal dynamics, enabling effective summarization of the video.

Process Breakdown :

1. **Feature Vector Processing**
    - As feature vectors pass through three successive neural network layers, they are refined and transformed at each stage to create progressively richer feature representations.
    - The final feature vector produced after the third layer encapsulates the most salient information about the input.
2. **Logit Conversion and Importance Scoring**
    - The final feature vector is passed through a model (such as a classification or regression head) that outputs a single value termed **logit**, representing the predicted importance of the corresponding frame.
    - This logit is then passed through a sigmoid function, which converts it into an **importance score** in the range [0, 1].
3. **Grouping into Segments**
    - Frames are grouped into segments based on the definitions provided in the ground truth data.
    - For each segment, the **average importance score** is calculated by aggregating the scores of all frames within the segment.
4. **Knapsack Algorithm for Segment Selection**

- Once segments and their average importance scores are determined, a **knapsack algorithm** is applied to select the most significant segments for the video summary.
- In this algorithm:
  - **Weights:** Represent the lengths of the segments.
  - **Values:** Represent the average importance scores of the segments.
- The knapsack algorithm optimally selects segments to maximize the total importance score while adhering to constraints, such as the maximum allowable summary length.

5. **Classification of Predicted Segments**
   - Segments included in the generated summary are classified as **1** (predicted as important), while those excluded are classified as **0** (not important).
6. **Evaluation Against User Ratings**
   - The binary predictions (1/0) for the segments are compared against the ground truth user ratings.
   - Evaluation metrics such as **F1-Score**, **Kendall's τ**, and **Spearman's ρ** are calculated to assess the model's performance:
     - **F1-Score:** Measures the balance between precision and recall.
     - **Kendall's τ:** Quantifies the rank correlation between predicted and ground truth rankings.
     - **Spearman's ρ:** Evaluates the strength and direction of association between predicted and actual segment scores.

This detailed process ensures that the summarization system not only identifies the most meaningful segments but also aligns closely with user preferences, achieving high-quality evaluation outcomes.

VideoSAGE Testing:

Preprocessing:

- Frame Downsampling: Videos were downsampled to **2 frames per second** (fps) for efficiency.

- Feature Extraction: Each frame's **features** were extracted using **GoogleNet's Pool5 layer** (feature size D=1024)
- Importance Scores: **Human-annotated** importance scores averaged from **15–20 evaluators per video.**

Graph Construction:

- For each video, we created graph representations where each node corresponds to a frame.
- Edges: Formed between **temporally adjacent frames** (up to **10 frames for TVSum, 20 frames for SumMe**), creating sparse graphs.

Training & Validation Setup:

- Data Splits: The datasets were divided into 5 splits, with 20% of videos used for validation and the rest for training.
- GNN Training: A lightweight Graph Neural Network (GNN) was trained for binary node classification (informative vs. non-informative frames).
- Epochs: Training was done for **40 epochs (SumMe)** and **50 epochs (TVSum)**, selecting the model from the epoch with the **lowest validation loss.**

Hyperparameter Tuning:
Used RayTune's grid search to find optimal hyperparameters:

- **Learning rate:** 0.001 (SumMe), 0.002 (TVSum)
- **Weight decay:** 0.003 (SumMe), 0.0001 (TVSum)
- **Batch size:** 1 for both datasets
- **Dropout rate:** 0.5 for both datasets

Performance Measurement :

F1-Score (F1):

- The F1-Score is the **harmonic mean of precision and recall**, used to measure the accuracy of the model. It ranges from 0 to 100%, with **higher values indicating better performance.**

Kendall's τ (τ):

- Kendall's τ is a correlation coefficient that **measures the ordinal association between two rankings.** A value of **1 indicates perfect correlation, and 0 indicates no correlation.** A negative value suggests an inverse relationship between the predicted and actual rankings.

Spearman's ρ (ρ):

- Spearman's ρ is another correlation coefficient, similar to Kendall's τ, used to measure the **strength and direction of association** between two ranked variables. Like Kendall's τ, **1.0 indicates perfect positive correlation, 0 indicates no correlation,** and negative values suggest inverse relationships.

TABLE I : TEST RESULTS

Predictions	F1-Score	Otani et al [20]*		Proposed Method	
	F1 (↑)	τ (↑)	ρ(↑)	τ(↑)	ρ(↑)
Random	54.37	0.00	0.00	−0.006	−0.009
Perfect	62.87	0.37	0.46	1.000	1.000

Random predictions result in a low F1-Score (54.37%) and no correlation with human rankings (τ = 0.00, ρ = 0.00). Perfect predictions achieve a higher F1-Score (62.87%) and perfect correlation (τ = 1.0, ρ = 1.0). The proposed method aims to improve correlation with human annotations, even if its F1-Score is slightly lower due to data segmentation.

Visual Validation of Performance :

SumMe/Video 1:

- PGL-SUM has a **higher F1-Score (69.69%)** than **VideoSAGE (64.43%),** meaning it is **slightly better** at predicting the most important frames.

- However, **VideoSAGE outperforms PGL-SUM in terms of both Kendall's τ (0.47 vs 0.08) and Spearman's ρ (0.60 vs 0.10),** indicating that VideoSAGE's predictions are **more aligned with human-annotated** importance scores and rankings.

TVSum/Video 16:

- PGL-SUM again has a **slightly higher F1-Score (68.54%) compared to VideoSAGE (62.16%)**, indicating better overall frame selection.
- However, VideoSAGE shows a similar Kendall's τ (0.31 vs 0.32) and a slightly lower Spearman's ρ (0.46 vs 0.47), which suggests that while **VideoSAGE might select slightly less optimal frames** based on F1-Score, its importance rankings are very close to the human annotations.

Conclusion:

PGL-SUM is slightly better in terms of F1-Score, but VideoSAGE outperforms it in terms of correlation metrics (τ and ρ), meaning that VideoSAGE provides better alignment with the human rankings of importance, even if its F1-Score is slightly lower.

TABLE II : EFFICIENCY AND PERFORMANCE OF VIDEOSAGE

Model	SumMe/**video_1**			TVSum/**video_16**		
	F1	τ	ρ	F1	τ	ρ
PGL-SUM*	**69.69**	0.08	0.10	**68.54**	**0.32**	**0.47**
VideoSAGE (ours)	64.43	**0.47**	**0.60**	62.16	0.31	0.46

Profiling Overview:

- All experiments (including PGL-SUM, A2Summ, and VideoSAGE) were run on a **Intel® Core™ i9-12900K processor** (3.2 GHz) with 32 GB of RAM.
- The evaluation was conducted using single-threaded execution to ensure fair comparisons across models.

- Profiling was done using PyTorch Profiler, with memory allocation details extracted via the export memory timeline command.

Interpretation

- **Inference Time:** VideoSAGE is significantly faster in terms of inference time (23.55 ms), outperforming PGL-SUM (113.79 ms) and A2Summ (120.59 ms) by a considerable margin.
- **Memory Consumption:** VideoSAGE requires much less memory (19.27 MB) compared to PGL-SUM (55.17 MB) and A2Summ (50.56 MB). In fact, VideoSAGE uses less than 2/5th of the memory used by PGL-SUM and A2Summ, making it much more efficient in terms of memory usage.

TABLE III : COMPARISON WITH OTHER SOTA MODELS

Model	Average Inference Time (ms)	Parameters' Memory (MB)	Max Memory Allocated (MB)
PGL-SUM*	113.79	36.02	55.17
A2Summ*	120.59	9.60	50.56
VideoSAGE (ours)	**23.55**	**3.52**	**19.27**

VideoSAGE not only outperforms existing methods in terms of inference speed but also demonstrates a significant reduction in memory usage, making it more efficient and scalable compared to state-of-the-art models like PGL-SUM and A2Summ

References and Links

[1] Jose M. Rojas Chaves, Subarna Tripathi, "VideoSAGE: Video Summarization with Graph Representation Learning" *arXiv.org*. https://arxiv.org/abs/2404.10539

[2] Github : https://github.com/IntelLabs/GraVi-T