Video Dialog

Lina LEKBOURI[1,2[72697]], Marc LICHTNER[1,2[72690]], and Hugo WERCK[1,2[72692]]

[1] NOVA School of Science and Technology (FCT NOVA), https://www.fct.unl.pt/
[2] TELECOM Nancy, Nancy, FRANCE, https://telecomnancy.univ-lorraine.fr/

1 Introduction

The primary objective of this project is to create a system that can understand and respond to natural language queries about video content, enabling users to find precise moments within videos without having to manually search through hours of footage.

Our work will be divided in three parts. In **Phase 1**, we establish the foundation by creating an **index of video moments** using Dual Encoders and OpenSearch.

Phase 2 extends our system by integrating multimodal encoders and decoders, enabling it to handle cross-modal queries and answer natural language questions about video content.

Our objectives of **Phase 3** will be to detect moments from a video. Indeed the user will enter the url of the video they want to detect the moment and in the end, they will get each moment description with their start and end timestamps.

Throughout this report, we document our **methodology, implementation details, challenges encountered, and the results of our experiments**. We also provide a critical analysis of our approach, discussing both its strengths and limitations, and suggesting possible improvements for future work.

2 Algorithms and Implementation

2.1 Embedding Representations

2.1.1 Embeddings neighborhood

The embeddings space provides a powerful mechanism to organize and retrieve data based on **semantic similarity** rather than exact keyword matches. To implement this capability in our video retrieval system, we extended our approach to include nearest neighbors search functionality using dense (knn) vector embeddings.

Revision of OpenSearch index

First, we create a specific field in our mapping to store the vector representations of each video caption (with a dimension of 384, which corresponds to the output dimension of our chosen embedding model):

```
{    "properties": {
        "caption_vec": {
            "type": "knn_vector",
            "dimension": 384,
            "index": True,
            "similarity": "cosine"
        }
    }
}
```

For generating the embeddings, we utilized the `sentence-transformers/all-MiniLM-L6-v2` model (from Hugging Face's Transformers library), designed for sentence embedding tasks.

We then processed all video captions in our dataset, computing their embedding vectors and storing them in a JSON file following the index mapping structure.

After generating all embeddings, we perform semantic searches taking an embedded query and searches the index for documents with embedding vectors closest to the query vector by using cosine similarity, which measures the cosine of the angle between two vectors.

To evaluate the effectiveness of our embedding-based search, we compiled a set of test queries representing different ways users might search for video content and then plot them to see their closeness. As an example:

Query	**Top Result Caption**
"A few other men are shown playing guitars as they sit."	"A few other men are shown playing guitars as they sit."
"A dog running in the park"	"man is doing a javeling throw in a big large field."
"A group of people in a meeting"	"The group plays for the audience, occasionally zooming in on individuals."
"A sunset over the ocean"	"woman used some black painting for make details, put the red stamp on the corner and finished the painting with yellow and reddetails on the flowers."

Table 1: Example queries and their top results using embedding-based search

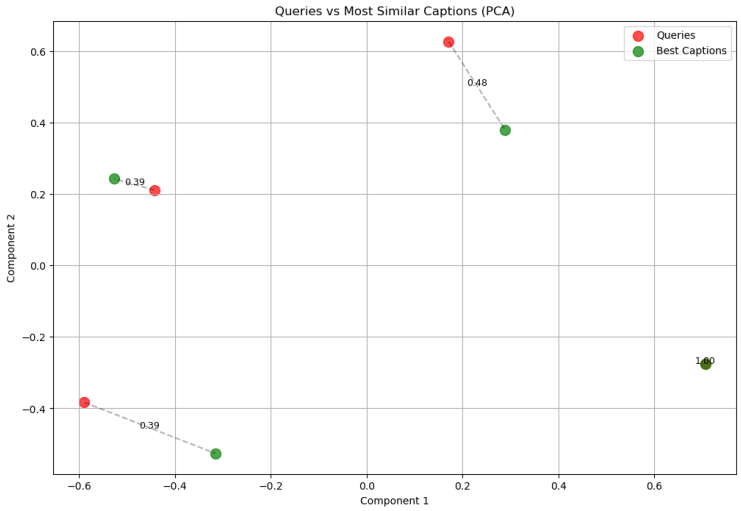

Fig. 1: Plot of Semantic search

The embedding-based search applied here, shows the cosine score between the query, its best match and their position on a 2D plan. We can clearly see that the query and their match are very close to each other, and clearly say that our result are much realistic. Another significant point is putting an exact caption as a query really find the good video, as shown by the point at the bottom right. Though it is costless to use text-base search in this case. In conclusion, by seeing this representation, we can assume that our encoder and our semantic search are doing their job as we wanted.

Indeed, our findings suggest that an optimal approach would combine both methods: using embedding-based search for capturing semantic relationships while retaining text-based search capability for situations requiring exact matching.

2.1.2 Constrained Embedding Searches

To achieve flexible and accurate video moment retrieval, several search strategies were implemented using OpenSearch and semantic embeddings:

- **Text-based search:** Classic keyword matching using the textual field `moment_description` in the index.
- **Embedding-based search:** Semantic retrieval using k-NN over Sentence-Transformer (`all-MiniLM-L6-v2`) embeddings.
- **Boolean filters:** Filtering by structured metadata such as video length.
- **Combined search:** Hybrid queries combining semantic similarity and boolean constraints (e.g., retrieving moments describing a specific action in videos longer than 5 minutes).

The OpenSearch index was custom-designed at the video moment level, including textual captions, embedding vectors, temporal bounds, and metadata. Experiments showed that focusing on a reduced set of well-chosen fields (e.g., caption embeddings and duration) improved retrieval precision and interpretability. Representative query example can be found in 2.1.1.

Empirically, semantic search using embeddings enables retrieval of relevant moments even for paraphrased or descriptive queries, while boolean constraints allow for precise filtering. Compared to pure text search, embedding-based retrieval is more robust to vocabulary mismatch and captures higher-level semantics.

2.1.3 Contextual Embeddings and Self-Attention

To better understand the power of transformer architectures in video retrieval, we analyzed the internal representations of language models and their attention mechanisms, using the `bert-base-uncased` model.

Contextual Embeddings. Here, we used the two following sentences as they have similar words used in slightly different context. Therefore showing the difference between static and contextual embeddings.
Sentence A: "Photosynthesis uses which energy?"
Sentence B:"Oxygen is released by photosynthesis, which uses the energy of sunlight."
At layer 0, the embedding we see is only static because the contextual embedding hasn't started yet. We see that words that appeared in both sentences are close even though they don't exactly mean the same thing as they are not used in the same context. At the last layers, we can see that those words were separated and respectively formed a cluster with their belonging sentence. For instance, the two words energy are extremely close on layer 0 and apart on layer 11. We can even guess that the word energy at the bottom belongs to the second sentence as it formed a cluster with the words "which," "release," "uses," and more in Annex A.1.1.

Positional Embeddings. We analyze a sequence of repeated tokens. Initially, the embeddings are almost identical, as they share the same meaning and lack contextual differentiation. However, as positional information mechanism come into play, the representations gradually diverge. This results in a smooth arc in PCA space, reflecting the incremental integration of context at each position, as shown in Annex A.1.2. Visualization of the pairwise distances between token embeddings is included in Annex A.1.2.

Self-Attention Mechanism. We examined attention matrices generated by the transformer. Visualizations reveal how the model distributes focus among tokens. Comparative analysis between cross-encoder and dual-encoder architectures showed that cross-encoders model token-level interactions, while dual-encoders favor efficiency. As layers and heads progress, attention becomes in-

creasingly focused and semantically precise, evolving from broad structural cues to task-specific understanding. Examples are provided in Annex A.1.3.

Interpretability. After obtaining an overview of the behavior of the attention heads as a function of the layers, we can distinguish three interesting cases in Annex A.1.4 :

- Two colored blocks and two dark blocks: Since the two sentences have been concatenated, each word in the first sentence is followed by those in the second. In this case, each word only pays attention to words in the same sentence, resulting in the presence of two main blocks along the diagonal of the attention matrix: one for each sentence. This symmetry around the diagonal represents the model's ability to recognize each word in its own context, without significant interaction between the two sentences.
- A diagonal shifted by one or more squares: Here, the attention matrix highlights that each word pays particular attention to one or more neighboring words. A shift of one square indicates that attention is primarily focused on the immediately following (or preceding) word, while a shift of several squares shows that attention is focused on more distant words. This reflects the model's ability to capture sequential or long-distance dependencies within the sentence.
- A vertical line: This pattern indicates that many words are paying particular attention to the same word, often the special token [SEP] that separates the two sentences. This means that the model uses this token as a cue or context summary, which can help in understanding the overall relationship between the two sentences after each word has established its own context.

2.2 Large Vision and Language Models

2.2.1 Cross-Modal Retrieval with CLIP

In this section, we focus on implementing and evaluating a cross-modal retrieval system using the CLIP model. The objective is to enable retrieval across different modalities —text-to-image, image-to-text, image-to-image, and text-to-text— by embedding all data into a shared semantic space.

The index must then be updated with key-frames extracted from videos, each associated with their captions or a generated description (using LLaVA, more details in part 2.2.3). Each frame and caption are encoded using CLIP encoder.

At query time, both text and image queries are encoded into the same vector space. A cosine similarity search is performed to retrieve the most semantically similar items—text queries retrieve frames or captions, and image queries retrieve similar frames or similar descriptions/captions.

We visualized the distribution of embeddings through 2D projections with PCA, distinguishing the queries and their top results. The plots for same-modality retrieval (text-to-text in Figure 12 in annexes and image-to-image in Figure 13) show strong clustering: the queries and retrieved elements are semantically aligned,

with results positioned close together. For example, querying a caption like "A woman is lifting some weights at the gym" successfully retrieves either the original or a very similar caption as display in Figure 14, and image queries return visually related frames, as shown in the example in Figure 15.

However, for cross-modal cases (text-to-image and image-to-text), results are more mixed. Although CLIP theoretically places all embeddings in the same space, the projections show partial separation between modalities. The top retrieved items still appear semantically relevant, but they are not always spatially adjacent on the plot, it is depicting in Figure 17 in annexes for text-to-image and in Figure 16 in annexes for image-to-text search. This might reflect subtle modality-specific biases or differences in how the model encodes fine-grained details across visual and textual inputs.

Even though the 2D representation raises this problem, the result emerging from the queries are still very impressive, by the similarity of the result it gave, for example in Figure 18 in annexes for text-to-image query or in Figure 19 that is a frame from one of the videos we are working with for image-to-text query.

To better interpret these distances, we unified all query-result pairs across all modalities into a single PCA graph in Figure 20 annexes. This allowed us to analyze how modality affects spatial distribution. While some overlap confirms the effectiveness of shared embedding space, the relative separation observed suggests that additional alignment techniques could further improve cross-modal cohesion view.

These observations support the conclusion that CLIP performs well in capturing semantic similarity across modalities, though interpretability and alignment may vary depending on modality and content complexity.

2.2.2 CLIP Interpretability

Language-Vision temporal similarity In this section, we computed the similarity between each frames and moments of a video using CLIP embeddings (as previous part). The goal was to plot the result for each moment so that we could visualize how the CLIP similarity score evolves overtime and ideally, observe peaks in similarity corresponding to the actual timestamp of each moment.

However, the resulting graphs didn't meet expectations as you can see in Figure 11. Instead of clear peaks, the similarity curves appeared irregular. This could be explained by two main factors:

1. **Repetitive Content in Videos**: The videos used in our dataset (mostly involving musicians, athletes, or artists) exhibit highly repetitive scenes. For instance, a caption such as "A man is playing the flute in front of a microphone" tends to receive high similarity scores across the entire video because multiple segments depict similar scenes. CLIP is not specifically trained on fine-grained domain distinctions (e.g., differentiating among musical instru-

ments or performance contexts) and as a result, struggles to distinguish between subtly different scenes involving similar actions.
2. **Recurring Visual Patterns**: The specific scene described in a given moment (e.g., the man playing the flute) often recurs multiple times throughout the video. Consequently, the same or visually similar frames appear before and after the moment's annotated timestamp. This leads to elevated similarity scores across a wider temporal range, making it difficult to pinpoint the exact moment based solely on the similarity plot.

2.2.3 LLaVA Integration

In order to provide visual question answering about videos, our system incorporates LLaVA. We created a search-based method to improve results that:

- Uses CLIP embeddings to find the most important video frames
- Puts these frames with the original query to LLaVA
- Gives answers based on what's in the image

To improve our search capabilities, we also employed LLaVA to produce descriptions for frames in the original dataset that lacked captions. Then, they were encoded and indexed to extent our capabilities of search.

There is an example on Figure 21. The image shows a man lifting two large weights above his head, with a smile on his face. He seems to be happy, enjoying the weightlifting session.

When we asked "What mood is in this scene?", LLaVA analyzed and responded correctly that the man seems in a positive mood. This anwser suits well with what we see on the image. It demonstrates LLaVA abilities to describe emotional context in a visual. It confirms its potential in interpretation.

2.3 Video moment detection

2.3.1 Captions generations

After having created a connection to OpenSearch, we download the required video and map accordingly the indices in the database. We extract 2 frames per second out of the video. Then, we generate the captions for every frames using LLaVA as described in 2.2.3, and update the corresponding descriptions in the database.

2.3.2 Frames and captions embeddings

At this point of the phase 3, we have an index with frame path, video ID and captions generated. To build upon the generated data, we employ the same methodology used in Phase 2. We use the CLIP model to create embeddings for both the extracted video frames and their corresponding captions. This step is essential for our pipeline as the resulting embedding vectors are important for the subsequent clustering stage, where we will group similar frames using

DBSCAN. Furthermore, the CLIP similarity score between frame and caption embeddings will be manipulated to identify the most representative caption for each detected moment.

2.3.3 DBSCAN clustering

Once we had the embedded caption of every frame, we wanted to use a clustering algorithm to link corresponding frames together. We looked for different techniques and finally opted for the dbscan algorithm, the key advantage was that we did not have to define the number of clusters beforehand and therefore led to better scaling according to the video. One source that we used for this decision was Reference 1.

The idea was to perform this algorithm with the embedded captions of the frames and then iterate on every cluster to compute the mean of the embedding of every captions related to the cluster. Finally, in order to compute the caption of the cluster, we initially wanted to decode the mean of the embeddings but we then opted to compute the cosine similarity between the mean and every caption to keep the most relevant. This was easier because we did not have a good decoder to go from the meaning of the embeddings to the semantics without proper training and fine tuning.

3 Evaluation

3.1 Dataset Description

For this project, the datasets we used are from ActivityNet. We had one file named `activity_net.v1-3.min.json`, which contained data about videos in the following format:

```
{video_id: {duration, subset, resolution, url, annotations: [{segment,
label}]}}
```

For each video in the file, we computed the number of annotations and kept the 10 videos with the highest count. Then, for each of them, we downloaded the video using the corresponding URL. We retained the `video_id`, `duration`, and `url`.

We then used three additional files from ActivityNet to retrieve the timestamps and captions for each moment in the videos. The format was as follows:

```
video_id: {duration, timestamps, sentences}
```

We were able to retrieve the data for each video using the `video_id`.

All the information for these 10 videos was stored in a variable named `top_videos`, structured as follows:

`{video_id, num_moments, url, duration, moments}`, where `moments` is a list of entries in the format: `{start, end, description, bag_of_words}`.

`start` and `end` represent the timestamps for the moment, and `bag_of_words` is the set of all words from the `description`.

Once we had parsed the JSON files into the `top_videos` structure as described above, we stored the data in OpenSearch. We indexed the data based on the moments, meaning that each entry in OpenSearch corresponds to one moment from a video.

Each document therefore contains the `video_id`, the `start` and `end` timestamps, the `moment_description`, and other information such as the `caption_vector` or `caption_bow`, which will be used later for search.

3.2 Baselines

To benchmark our video moment retrieval pipeline, we compare against three reference methods implemented in Phases 1 and 2:

- **Text-based Search**: a bag-of-words approach that performs exact keyword matching on the `moment_description` field. Retrieval quality is evaluated by the cosine similarity between the query and the best-matching description. The overall similarity distribution and example top-1 results are shown in Figure 1 and Table 1.
- **Embedding-based Search**: semantic retrieval using Sentence-Transformer embeddings (`all-MiniLM-L6-v2`) indexed with k-NN. Moments are ranked by cosine similarity in the 384-dimensional embedding space. A 2D PCA projection (Figure 1) and qualitative examples (Table 1) illustrate the closeness of queries to their top matches.
- **CLIP-based Retrieval**: cross-modal retrieval leveraging CLIP's joint text–image embedding space. Both text queries and video keyframes are encoded by CLIP, and ranking is based on their cosine similarity. The temporal evolution of similarity scores for each ground-truth moment is plotted in Figure 11.

These baselines provide solid reference points for evaluating our Phase 3 moment detection method. As shown by the quantitative and qualitative analyses above, CLIP-based Retrieval outperforms the purely textual and embedding-only approaches, making it the best-performing baseline in our setup.

3.3 Results analysis

3.3.1 Language-Vision Contrastive Moments

Contrastive Similarity Matrix. For each video, we computed a matrix. Element (i, j) represents the CLIP similarity score between (keyframe moment i, caption moment j). Looking at the matrix on Figure 22 draws our attention to:

- **High Similarity Scores (Yellow):** Frames and captions with high similarity scores, such as Frame 2 and Caption 2, indicating a good match between the visual content of the frames and the description of the captions. We can interpret that the visual elements in these frames are well-represented by their captions.

– **Low Similarity Scores (Purple):** Frames and captions with low similarity scores, such as Frame 1 and Caption 4, show a mismatch between visual and textual aspects. This could be due to the captions describing elements not present in the frames or the frames containing visual elements not captured by the captions.

Interpretable Visual Maps. We further analyzed which regions in each frame contribute most to the similarity scores by generating attention heatmaps for all frame-caption pairs, see Figure 23:

– **High Relevance Areas (Red):** For example with Frame 2, Caption 2, we might see the regions within the frames that are most relevant to the captions. These areas enable us to see the visual elements that are mostly contributing to the similarity scores.
– **Low Relevance Areas (Blue):** With Frame 1, Caption 4, we see the regions within the frames that are the least important for the captions. These areas suggest that the visual elements in these regions do not suit very well with the caption descriptions.

3.3.2 Video Moment detection An example of PCA projection of the clustering can be found here: Figure 24. As depicted in this figure of the CLIP-captions embeddings, we can see that roughly at least a third of the points are labelled as noise. However, mapping the cluster labels back to the timeline reveals that the algorithm isolates three semantically coherent segments:

– **Moment 1 (00:25–00:47)** – *"A person on a jetski creating waves as they go around the corner of a bay."*.
– **Moment 2 (00:26–03:06)** – *"A person on a surfboard riding a wave."*.
– **Moment 3 (01:33–02:31)** – *"the ocean with white foamy water and waves rolling in."*.

4 Critical discussion

4.1 Phase 1 — Cross-modal search baseline

From phase 1, we were able to make queries with frames and texts to retrieve similar frames or captions. Eventhough we didn't use directly this feature on phase 3, this phase was critical for us to get hands on practice with OpeanSearch, use CLIP to encode our embeddings, compute the attention and then interpret the result from the different heads and layers with the attention matrix. This part seems now very easy to implement and to analyze but at the time, because there were a lot of new notions we felt lost on what we really needed to do, we also encountered a lot of problems with OpenSearch as the API endpoint was unacessible almost everytime we wanted to work on the project. At the end of this first task, we were able to use OpenSearch, to understand how and why to use CLIP and we were able to visualize how precise our program was in order to retrieve the most relevant frame or caption.

4.2 Phase 2 — Moment-level indexing

In phase 2, we focused on the different moments of the downloaded videos. We rearranged the OpenSearch index accordingly so that one entry is now a moment and a frame instead of the entire video. We captioned moments with LLaVA-3.0 and inserted both captions and CLIP vectors into OpenSearch. However, the result we had was not as expected in class because we were supposed to guess the timestamp of the different moments by plotting a matrix showing the cosine similarity between the moment caption embedding and the frame embedding over time which was in fact not the case.

4.3 Phase 3 — Zero-shot moment detection

For the final phase, we implemented a zero-shot moment detection and generated a corresponding caption for each of them. As you can see in the result, a lot of frames are not related to any moment (considered as noise by DBMSCAN) and because we didn't use any "temporal embedding" so that close frames are more likely to be in the same moment, we can have frames in the beginning and the end of the video together. Another approach could have been to iterate over every frames that are initially marked as "unprocessed" at the beginning. Go to the first unprocessed frame, create a new moment and to add all the next frames that have a cosine similarity between the two captions that are above a defined threshold and stop the moment when a frame is below this threshold (we can also imagine a system of patience that allow few frames to be below that threshold before stopping the moment so that a single bad frame doesn't stop a moment that would have continued without this specific frame). Then, we mark all the frames we went through as "processed" and iterate until all frame have been analyzed. Therefore, depending on the usage of the zero-shot moment detection we need, we can adapt the implementation so that it fits better the requirement. From one side, we can have long moments in which the frames are not necessary really related to the moment itself but with more flexibility and a decent number of moments. On the other hand, using the technique describe above, we would have a more precise timestamp for every moment but with possibly an explosion of number of moments and perhaps multiple time the same kind of moments during different part of the video.

Such a tool could be very useful for web scraping of videos on platform like Youtube. Nowadays, Youtube is a gold mine of information and can be used to gain a good knowledge or as guide to develop various skills, however it can be hard to find a good video or a video that really corresponds to a very specific subject we are looking for. Therefore, it can be hard and a long process to find what we need. In this context, this tool can be use on larger scale to download and index a lot of videos, to detect the different moments and to annotate them so that if we are looking for something specific we can directly go to the relevant timestamp of the moment without having to watch the entire video to be sure that we did not missed the relevant part.

5 References

1. **DBScan**: https://www.newhorizons.com/resources/blog/dbscan-vs-kmeans-a-guide-in-python
2. **MPDW course lectures 2024-2025 at Nova FCT**: https://wooded-guitar-ad7.notion.site/Web-Search-and-Data-Mining-1b49432046bc8181b61cf93d5c865d99

A Annexes

A.1 Phase 1

A.1.1 Contextual Embeddings Visualizations : provides visualizations of contextual embeddings across transformer layers.

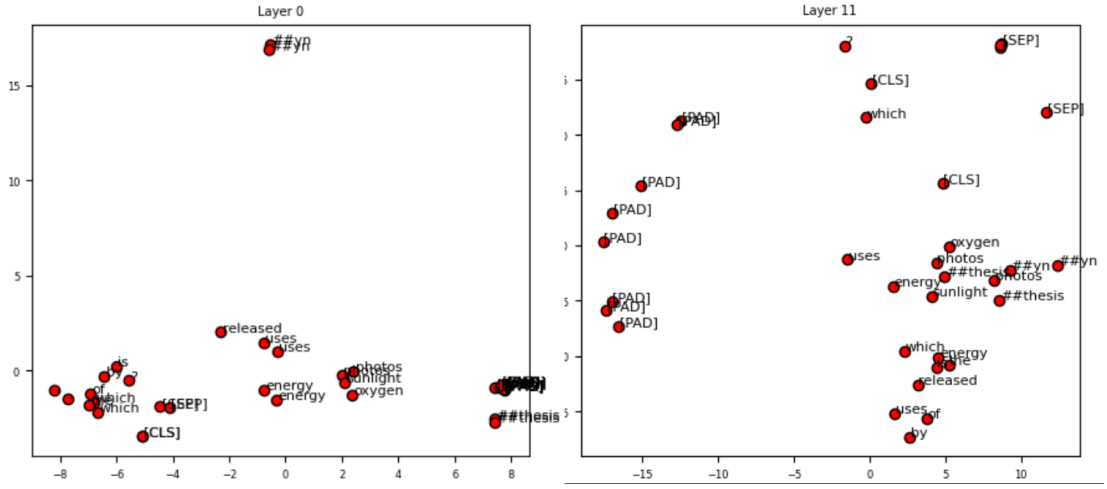

Fig. 2: PCA visualization of token embeddings of layers 0 and 11 (`bert-base-uncased`).

A.1.2 Positional Embeddings : shows the pairwise distances between positional embeddings for repeated tokens, highlighting how positional information is encoded even for identical input tokens.

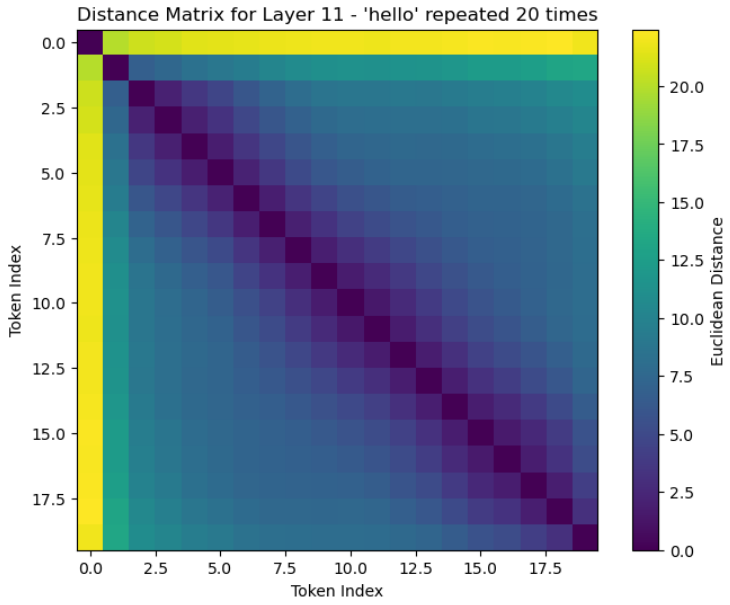

Fig. 3: Pairwise distances between positional embeddings for repeated tokens.

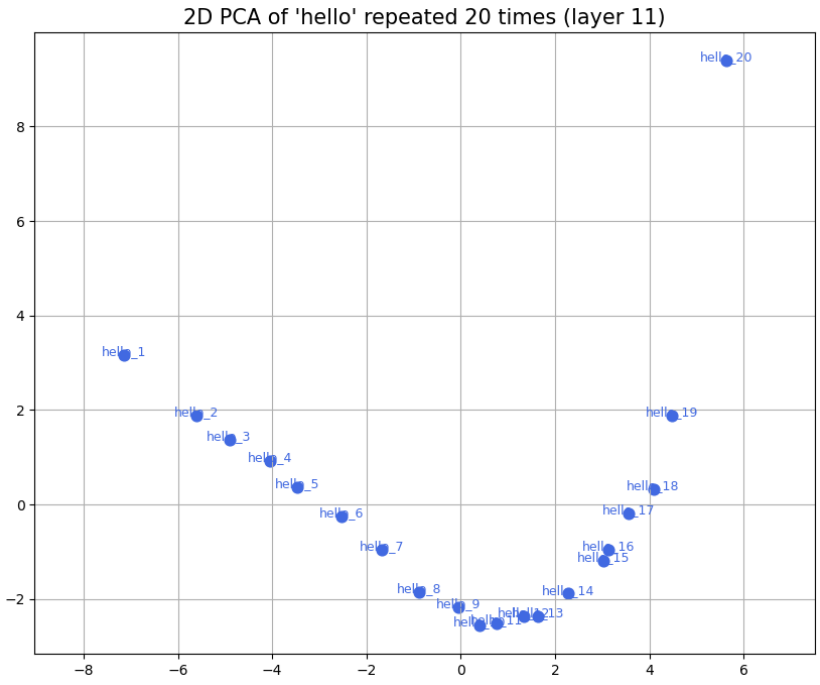

Fig. 4: Embeddings plot for repeated tokens.

A.1.3 Self-Attention Maps : presents self-attention matrices for both cross-encoder and dual-encoder architectures.

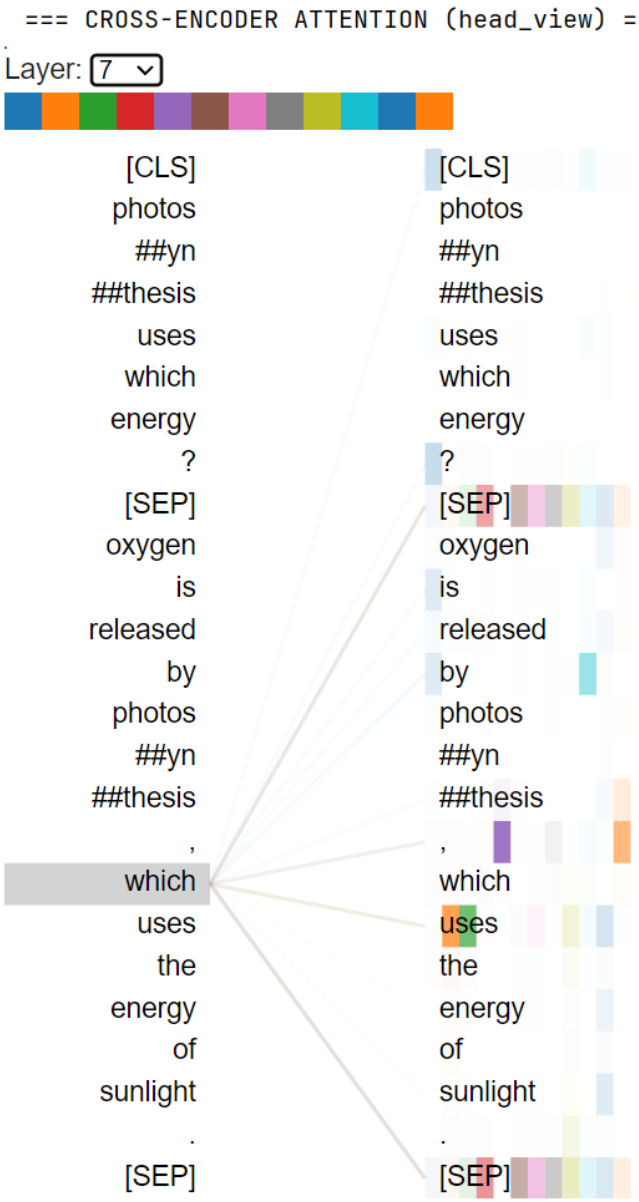

Fig. 5: Self-attention matrix for a cross-encoder, word 'which' at layer 7 (`bert-base-uncased`).

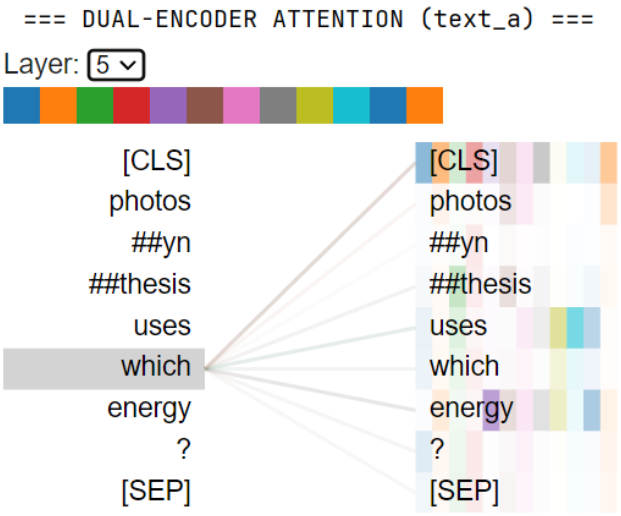

Fig. 6: Self-attention matrix for a dual-encoder, layer 5 from text a (`all-MiniLM-L6-v2`).

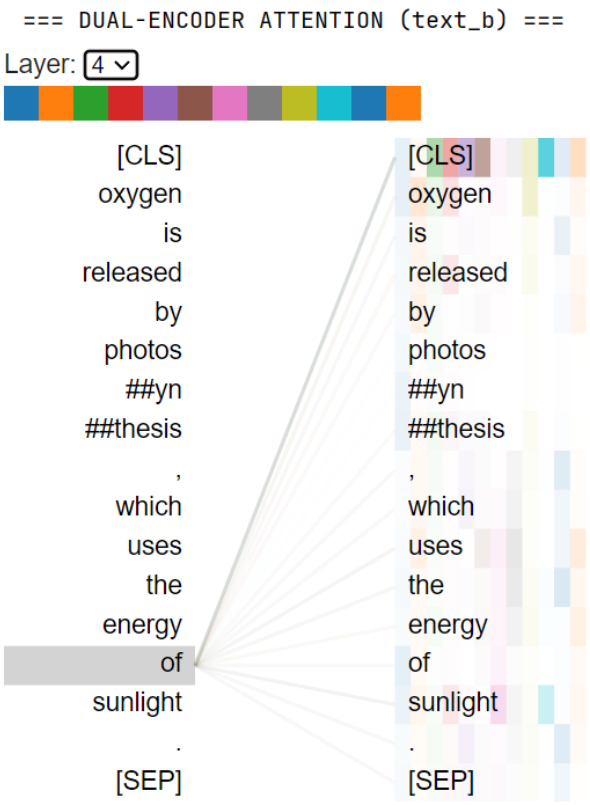

Fig. 7: Self-attention matrix for a dual-encoder, layer 4 from text b (`all-MiniLM-L6-v2`).

A.1.4 Interpretability. :

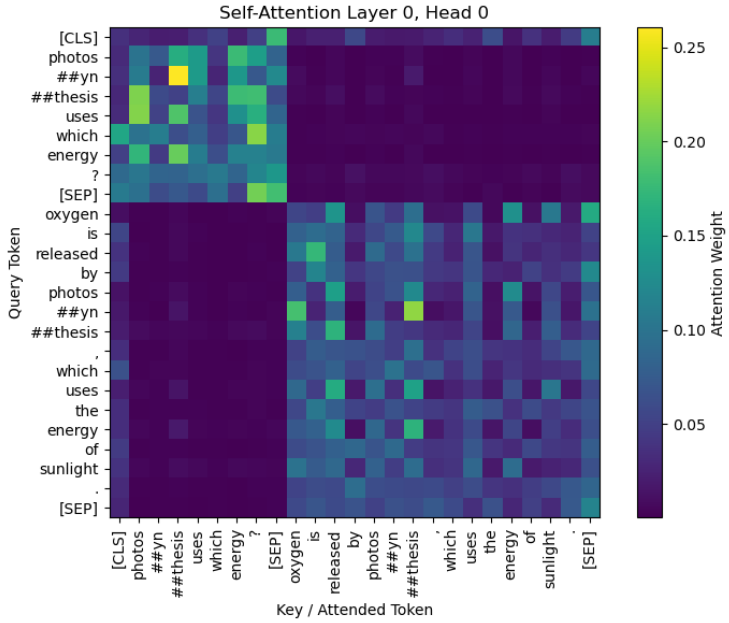

Fig. 8: Layer 0, Head 0

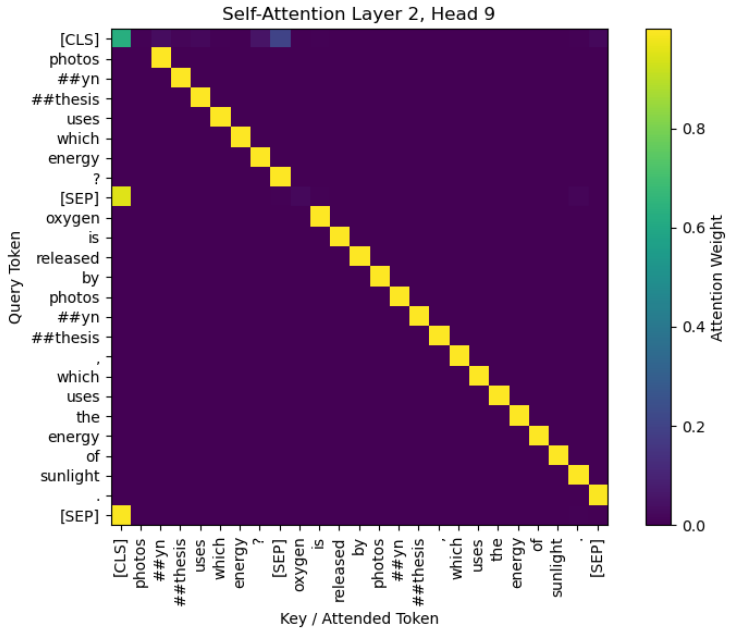

Fig. 9: Layer 2, Head 9

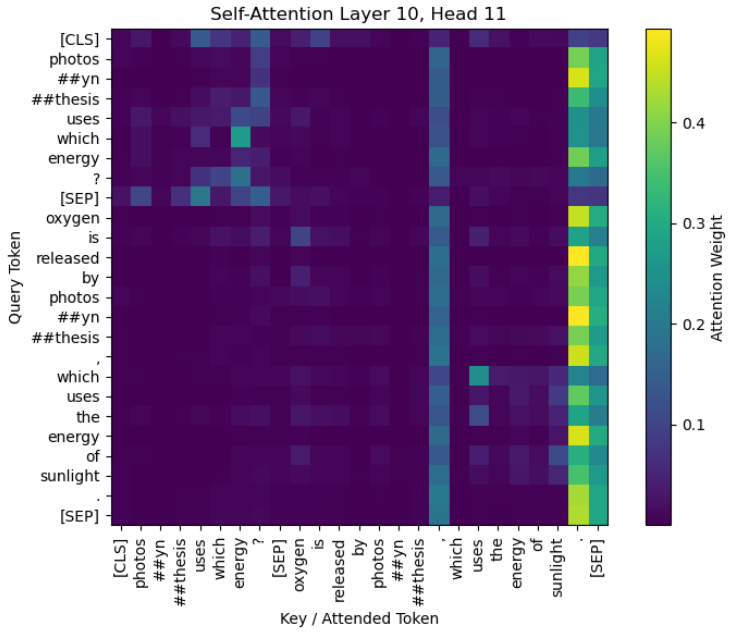

Fig. 10: Layer 10, Head 11

A.2 Phase 2

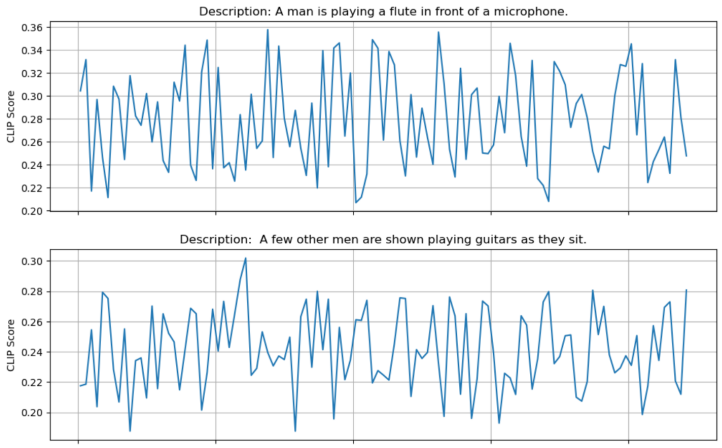

Fig. 11: CLIP similarity between frames and moments

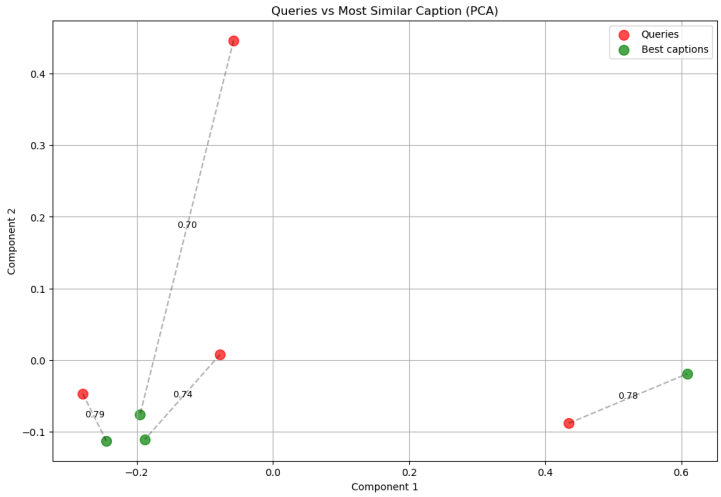

Fig. 12: Text queries and their closest text results using CLIP encoder

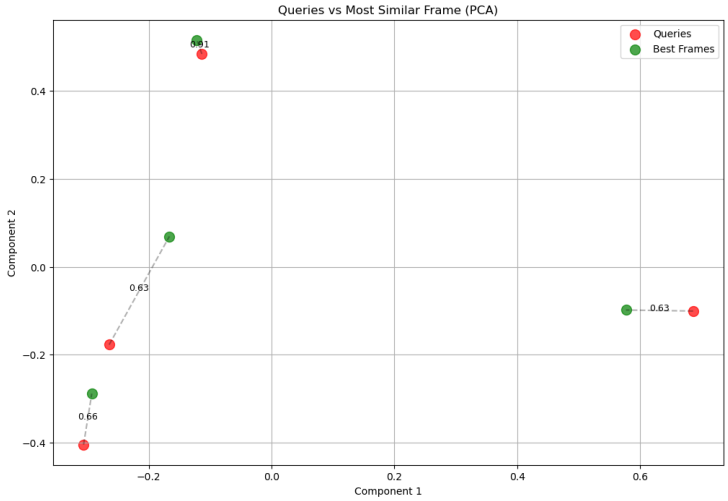

Fig. 13: Images queries and their closest image results using CLIP encoder

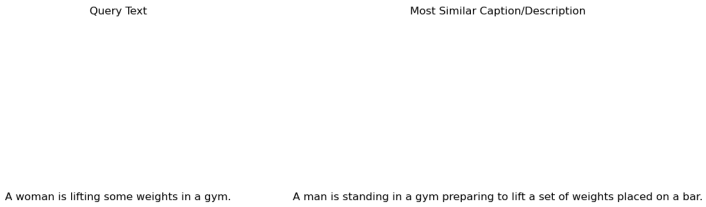

Fig. 14: Example of text query and its closest text result

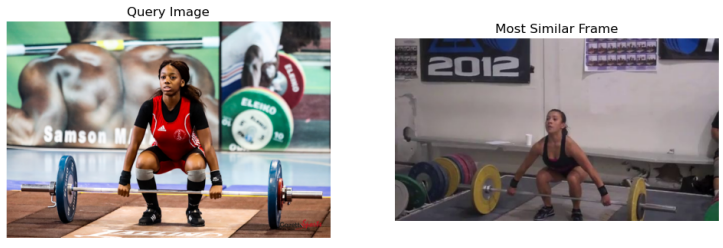

Fig. 15: Example of image query and its closest image result (frame)

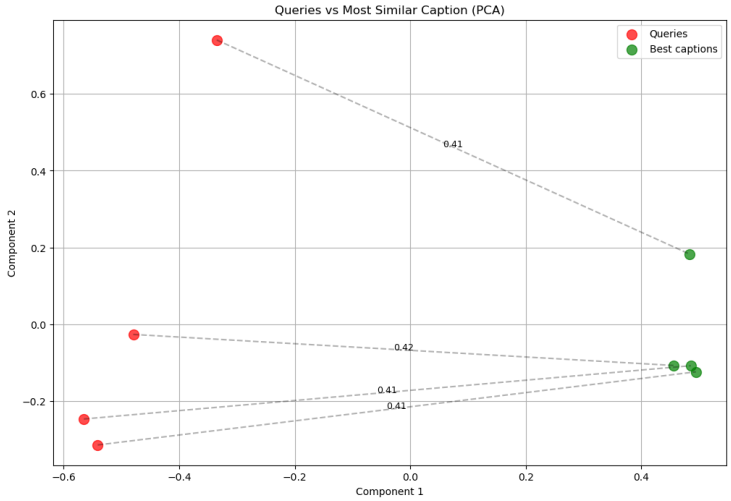

Fig. 16: Images queries and their closest text results using CLIP encoder

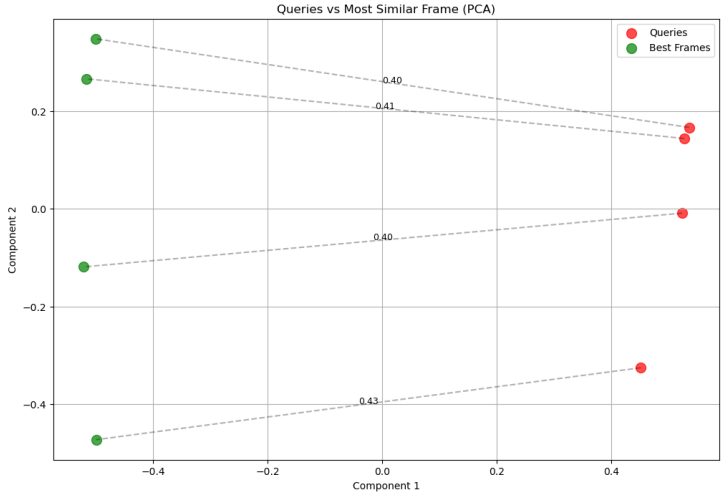

Fig. 17: Text queries and their closest image results using CLIP encoder

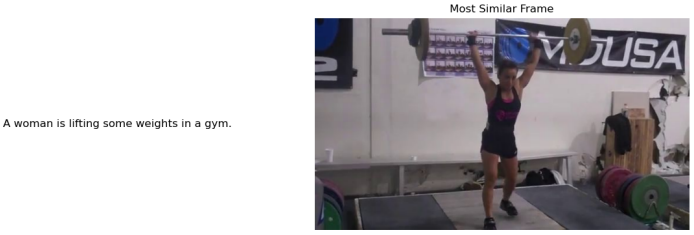

Fig. 18: Example of text query and its closest image result (frame)

man is standing inside a court practicing a shot put.

Fig. 19: Example of image query and its closest text result (description or caption)

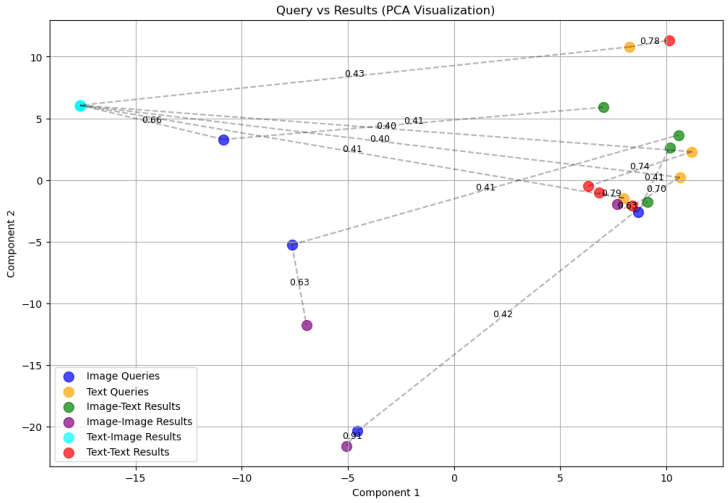

Fig. 20: Both text and image queries and their text or image closest result using CLIP's encoder

Fig. 21: Weightlifting Moment: Lifting Barbell with Positive Mood

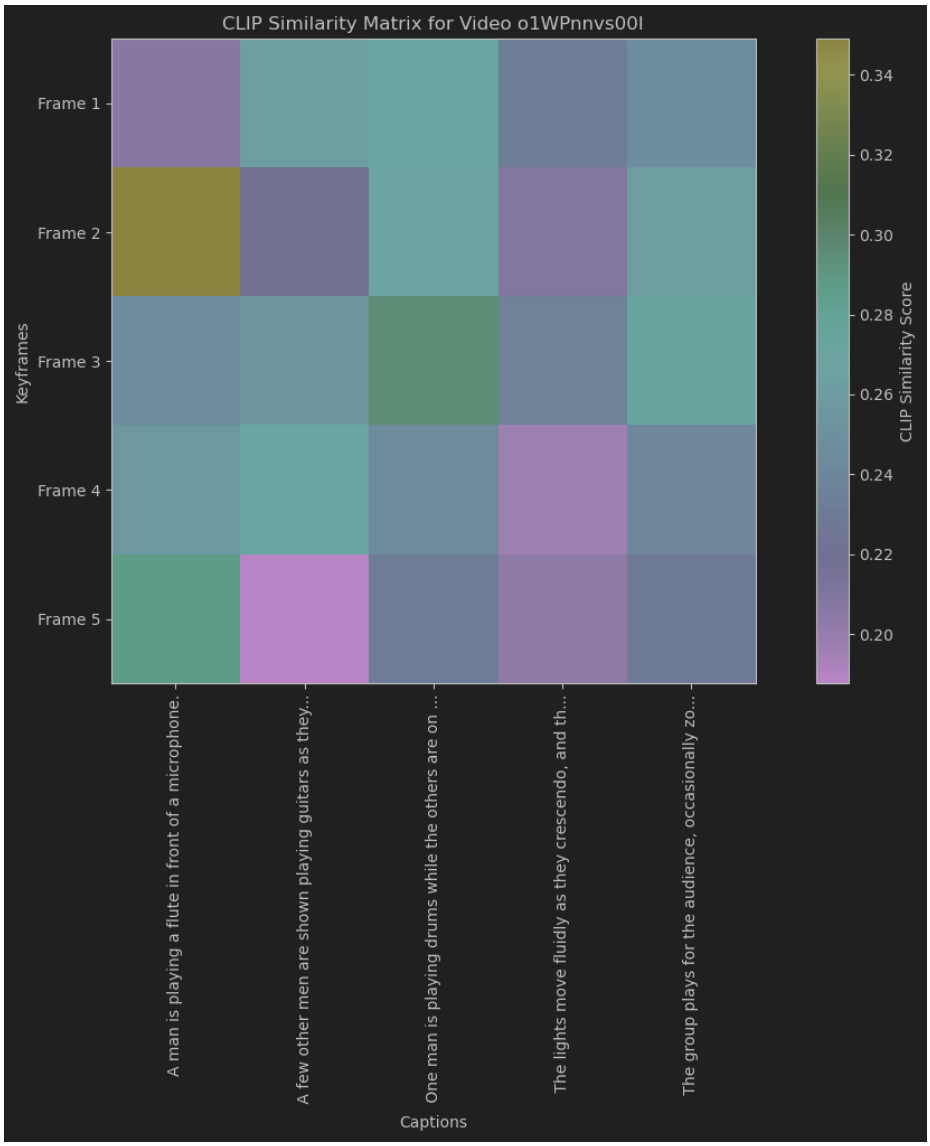

Fig. 22: CLIP similarity matrix between keyframes and captions for video o1WPnnvs00I

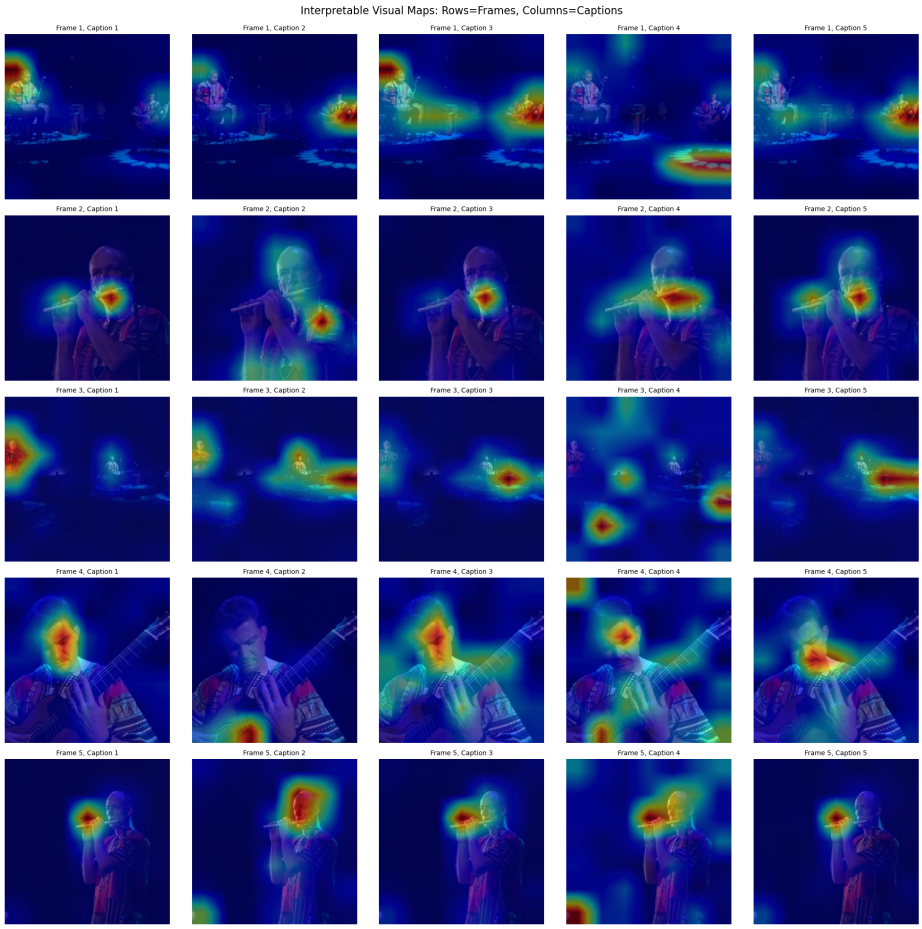

Fig. 23: Interpretable Visual Maps: Rows=Frames, Columns=Captions

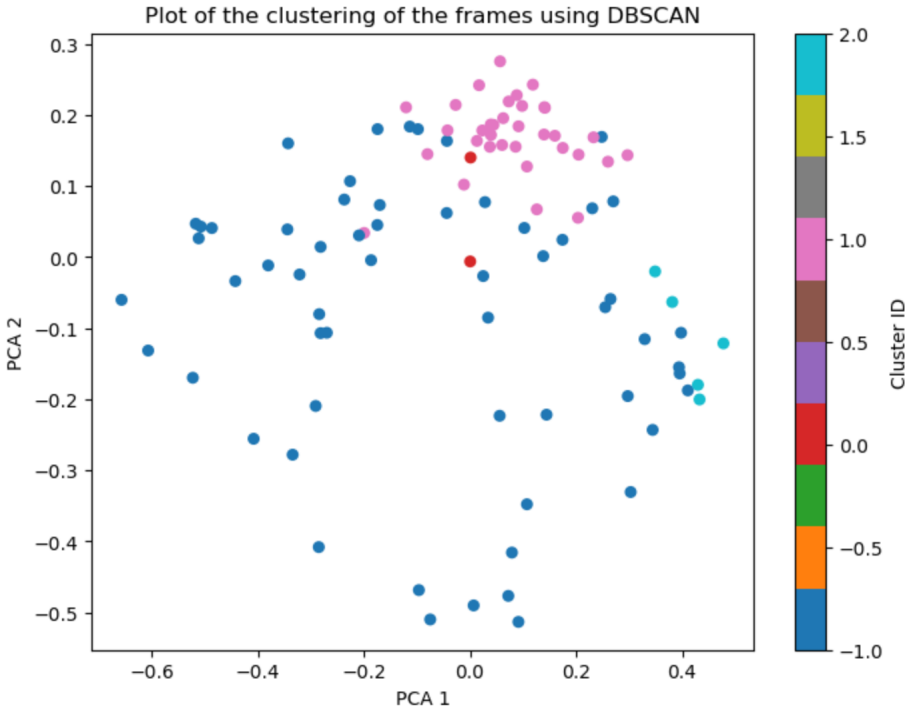

Fig. 24: DBSCAN clustering on the embedded captions