**Link to the video Demo :** [**https://f.io/GnbFanve**](https://f.io/GnbFanve)

**Title -Video Summarization working**

**Part 1**: Video Summarization using [Intel branch](https://github.com/srinarayan-srikanthan/Video-LLaVA/tree/intel) of open source [video-llava](https://github.com/PKU-YuanGroup/Video-LLaVA/tree/main)

1. Git clone <https://github.com/srinarayan-srikanthan/Video-LLaVA.git>
   1. cd video-llava
   2. conda env create -f environment.yaml
   3. conda activate llava
   4. git checkout intel
   5. git apply my\_changes.patch
   6. Run inference from command line:
      1. python -m videollava.serve.cli --model-path LanguageBind/Video-LLaVA-7B --file <PATH\_TO\_VIDEO> --device cpu --mode ipex

* Steps to make sure it is working.
  + Debug: If there are troubles installing specific versions of torch/torchvision/torchaudio, use this command to install from source, and then re-run conda env create -f environment.yaml:
    - pip install torch==2.1.1+cpu torchvision==0.16.1+cpu torchaudio==2.1.1+cpu --index-url https://download.pytorch.org/whl/cpu
* Changes to split Video into 2.5 sec chunks.
  + Once model is loaded into memory, video files located locally on disk (presumably saved once recorded from CCTV camera) are read into a processing function. This function uses python libraries to trim video files into 2.5 second chunks, and systematically passes them to the LVM alongside a query to ‘Describe the video’. This method allows us to get a play-by-play like output generated from the model.
  + There is no definitive rule stating that videos must be segmented into 2.5-second chunks for processing. Depending on the specific requirements of your use case, you may choose to implement a more granular approach. Python code can be utilized to segment videos, extract a specified number of frames, and subsequently feed them into the model for processing.
* Changes Made to optimize video lama on Xeon
  + **Covered in the patch file, additionally run this file once the environment is setup to get all the h/w related tunable parameters set** [**https://github.com/intel/intel-extension-for-pytorch/blob/main/examples/cpu/inference/python/llm/tools/env\_activate.sh**](https://github.com/intel/intel-extension-for-pytorch/blob/main/examples/cpu/inference/python/llm/tools/env_activate.sh)
  + Eg: wget [**https://github.com/intel/intel-extension-for-pytorch/blob/main/examples/cpu/inference/python/llm/tools/env\_activate.sh**](https://github.com/intel/intel-extension-for-pytorch/blob/main/examples/cpu/inference/python/llm/tools/env_activate.sh)
  + source env\_activate.sh

**Getting Visual Rag Working**

* How to download example open vino LLM Rag implementation **(Avinash to update steps)**

Clone this sample from OpenVINO :https://github.com/dhandhalyabhavik/model\_server/tree/visual-rag/demos/python\_demos/visual\_rag

* How to make changes to store clip embeddings
  + The visual RAG has 2 main components,
    1. clip embeddings generation and storing it into DB
       - Run either chroma or VDMS docker. For chroma use (docker run -d -p 8000:8000 chromadb/chroma), for vdms use (docker run -d -p 55555:55555 intellabs/vdms:latest)
       - Run this file python3 generate\_store\_embeddings.py
    2. Retrieval of frame or video based on user’s prompt
       - Run image retrieval ui using command streamlit run image\_retrieval\_ui.py --server.address 0.0.0.0 --server.port 50022
       - Run video retrieval followed by model assisted answer generation using command streamlit run visual-rag-ui.py --server.address 0.0.0.0 --server.port 50055

Detailed steps are described in following git repo https://github.com/dhandhalyabhavik/model\_server/tree/visual-rag/demos/python\_demos/visual\_rag