NYU

CS-GY CS-UY 4613 6613 2025: Artificial Intelligence

Chat With Your Video Library

Background

Retrieval-Augmented Generation (RAG) systems combine retrieval-based methods with generative models to deliver accurate, context-aware responses grounded in external data sources. Unlike traditional chatbots limited by static knowledge, RAG systems dynamically fetch relevant information and generate coherent answers.

This project focuses on building a course-specific chatbot that leverages lecture videos to answer student queries. The system will retrieve relevant video segments and generate responses tied directly to those clips, offering a multimodal learning experience. Emphasis is placed on understanding and implementing core RAG components without relying on high-level frameworks.

Objectives

The objective is to design and deploy a functional RAG-based chatbot that answers course-related questions by:

1. **Building a Data Pipeline** to stream, process, and store video frames aligned with subtitles in MongoDB.
2. **Fine Tuning a Vision-Language Model (VLM)** for domain-specific understanding.
3. **Implementing Featurization** to convert video-subtitle data into vector embeddings stored in Qdrant.
4. **Developing a Retrieval System** to fetch relevant video clips based on user queries.
5. **Deploying an Interactive Chatbot** using Gradio and Gemini for real-time, grounded responses.
6. **Demonstrating Capabilities** through a set of predefined queries with linked video segments.

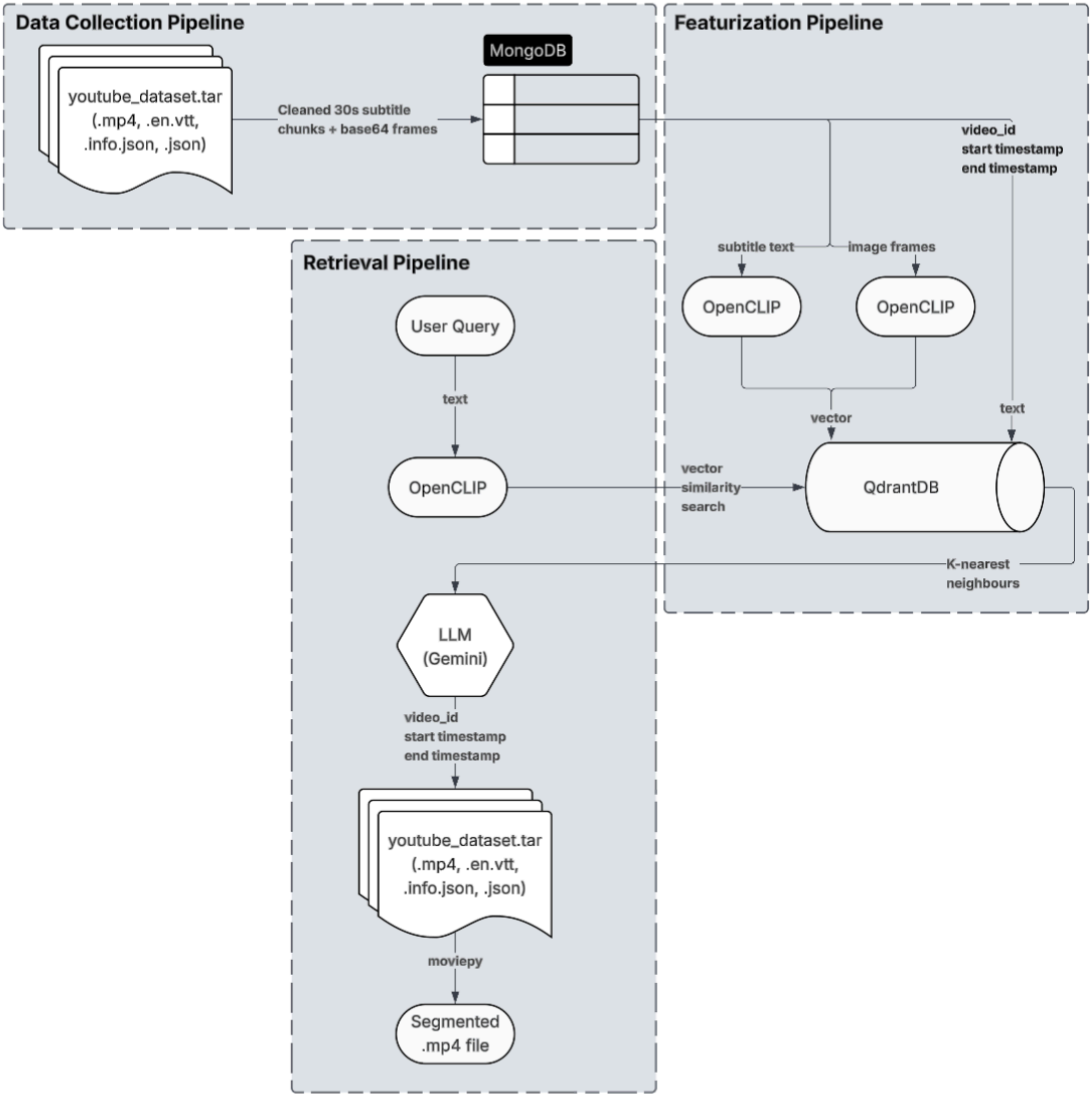
The end goal is a modular, scalable RAG system that enhances user interaction by combining AI-generated answers with precise video references.

CS-GY CS-UY 4613 6613 2025: Artificial Intelligence

Team Members

NAME	NetIDs
Swapnil Sharma	ss19753
Sindhu Jyoti Dutta	sd6201

Workflow



Data Collection pipeline:

The pipeline starts by importing essential libraries for dataset streaming, subtitle parsing, video and image processing, and MongoDB operations. After setting up the environment, a connection to MongoDB is established using credentials from environment variables. The target database and collection are selected, and any existing documents are cleared to ensure fresh data insertion.

Next, the video dataset, stored in webdataset format, is streamed using HuggingFace's load_dataset with streaming mode enabled to handle large files efficiently. The pipeline iterates through each sample in the dataset.

For every sample, it first checks for the presence of the required files: subtitles (en.vtt), metadata (info.json), and video (.mp4). If any of these files are missing, the sample is skipped to maintain data consistency.

Once validated, the subtitle file is decoded into a string, and the metadata is parsed to extract important identifiers like video_id and title. The video content is saved temporarily on disk to allow frame extraction.

The subtitles are then processed using a fixed window chunking method. Captions are grouped into 30-second intervals, and redundant text or filler words are removed for clarity. For each chunk, a representative frame is extracted from the midpoint of the time window and encoded in base64 format.

Each chunk, containing cleaned subtitle text, start and end timestamps, and the base64-encoded frame, is enriched with metadata such as the video ID and title. These chunks are then inserted into the MongoDB collection as structured documents.

After processing, the temporary video file is deleted to manage storage efficiently. Any errors encountered during processing are logged, and the pipeline continues with the next sample, ensuring smooth and uninterrupted data collection.

Featurization Pipeline Milestone

This featurization pipeline is designed to convert video subtitle chunks and their corresponding frames into multimodal vector embeddings and store them in a Qdrant vector database for efficient retrieval.

The process starts by importing libraries required for deep learning, image processing, database operations, and vector storage. Once the environment is ready, the pipeline connects to MongoDB to access the previously stored video-subtitle chunks.

The next step is to **load a pre-trained OpenCLIP model** (ViT-B-32 variant) from the open_clip library. The model is initialized on GPU if available, otherwise on CPU. Along with the model, the appropriate tokenizer and preprocessing transforms for images are also loaded.

Two embedding functions are defined:

- embed_text(text): Tokenizes the input text and uses OpenCLIP to generate a text embedding vector.
- embed_image_from_base64(b64): Decodes a base64 image, preprocesses it, and generates an image embedding vector.

After setting up the model and embedding functions, the pipeline **retrieves documents from MongoDB**. For each document containing both subtitle text and a frame:

- It generates a **text embedding** from the cleaned subtitle text.
- It generates an **image embedding** from the associated video frame.
- A **multimodal vector** is created by combining both embeddings, giving 70% weight to the text and 30% to the image, ensuring that semantic meaning from the subtitles is prioritized.

Each multimodal vector, along with metadata (like video ID, title, timestamps, and file path), is structured into a PointStruct object and added to a list for batch insertion.

Once all valid documents are processed, the pipeline connects to the Qdrant vector database. It recreates the target collection (video_chunks_multimodal) with a cosine similarity configuration, ensuring previous data is cleared.

Finally, the pipeline **uploads all multimodal vectors** into Qdrant using the upsert method, making them available for fast similarity-based retrieval in downstream tasks.

If no valid data points are found during processing, the pipeline gracefully exits with a message.

CS-GY CS-UY 4613 6613 2025: Artificial Intelligence

Retrieval Milestone

This pipeline implements the core retrieval, reasoning, and response generation flow for the Video-based RAG system. It allows users to input natural language questions and receive answers grounded in relevant video segments, along with the corresponding video clip.

The process begins by loading a **pre-trained OpenCLIP model** (ViT-B-32), which is used to convert user queries into vector embeddings. The model is initialized on GPU if available, ensuring efficient computation.

A connection is established with the **Qdrant vector database**, which stores multimodal embeddings of video subtitles and frames. When a user submits a question, the pipeline embeds the query text and performs a similarity search in Qdrant to retrieve the top 15 most relevant video chunks based on cosine similarity.

The retrieved chunks are then **formatted into a contextual block**, where related segments from the same video are merged if they are temporally close. This ensures that the context provided for reasoning is coherent and consolidated.

Next, the structured context and the user’s question are passed to **Google's Gemini model**. A carefully crafted prompt instructs Gemini to generate an answer strictly using the provided context. The model also outputs metadata, including the video ID, title, and precise timestamps for the relevant segment.

Once the answer is generated, the pipeline **extracts metadata** from Gemini's response using regex parsing. It identifies the video file and calculates the exact start and end times in seconds.

Using this metadata, the pipeline locates the corresponding video file and employs moviepy to **extract the relevant video segment**. The clip is saved temporarily for display.

Finally, a **Gradio interface** is launched, providing a simple frontend where users can input questions. The system returns the generated answer along with the extracted video clip, offering a seamless multimodal interaction grounded in actual course video content.

This pipeline effectively integrates embedding-based retrieval, LLM-driven reasoning, and dynamic video processing to deliver a robust Video RAG application.

Other Approaches

Frame Variance Chunking

One strategy involved calculating a hash for each extracted video frame and comparing consecutive frames to detect changes. If two frames were highly similar, the chunk was skipped to minimize redundancy in the dataset. While this reduced duplicate information, it led to extremely small and fragmented chunks — often spanning just a few seconds. Since much of the video's meaning was carried through the subtitles rather than visual variance, the resulting chunks lacked sufficient textual content for meaningful retrieval. This significantly degraded the quality of query responses, as the transformer model struggled to infer context from tiny, visually-driven snippets with minimal subtitle support.

Subtitle Variance Chunking

To better leverage the richness of subtitles, another method compared consecutive subtitle entries and skipped segments if they appeared redundant. Although this reduced duplication more effectively than frame hashing, it introduced a new problem: many chunks became too short, often only 2–3 seconds long, containing merely 5–6 words. Such brief segments did not provide enough context for the transformer model to understand and reason over user queries. As a result, retrieval performance suffered, and answers generated were often shallow or lacked continuity, showing that over-aggressive redundancy removal hurt downstream comprehension.

Topic Modeling Chunking

An attempt was also made to use BERTopic, a transformer-based topic modeling method, to segment the videos. The goal was to group subtitles and frames into semantically coherent topics, providing more meaningful chunk boundaries. While BERTopic successfully assigned different topic IDs to distinguish sections of the video, it did not significantly improve overall retrieval quality. **This was likely because topic modeling focuses on global coherence across documents, rather than preserving fine-grained temporal alignment critical for precise video retrieval**. However, BERTopic proved valuable as a validation tool, verifying that the transformer chunks/embeddings captured topic shifts appropriately across the lecture videos.

Fixed 1 Minute Chunking

To simplify the chunking process, a fixed window strategy was tested, initially using 1-minute intervals. This approach ensured that every chunk contained enough subtitle and visual context for robust retrieval. However, the downside was that 1-minute chunks included too much information, often spanning multiple unrelated subtopics. When retrieved during querying, the resulting video clips were unnecessarily long, diluting answer precision. Therefore, the chunk duration was reduced to 30 seconds, striking a better balance between context richness and precision in the final video segments returned to users.

Fine Tuning K in K Nearest Neighbors (KNN)

Another important tuning step involved adjusting the number of nearest neighbors (K) during retrieval. Initially, higher K values retrieved too many loosely relevant chunks, making it harder for the transformer to generate focused answers. After experimentation, setting K = 15 provided the best results: it offered a manageable pool of 5–10 strong candidates for the model to reason over without overwhelming it. This fine-tuning step helped ensure that the generated video responses were more focused, shorter in length, and better aligned with user queries.

Lessons Learned

1. **Context Window Size Matters:** Transformer-based models require a minimum context size (around 20–30 meaningful tokens) to generate stable embeddings. Chunks that were too small, especially from frame and subtitle variance methods, degraded model reasoning and retrieval quality.

2. **Multimodal Fusion Weighting Impacts Retrieval Quality:**
Giving 70% weight to subtitle (text) embeddings and 30% to frame (image) embeddings during fusion produced stronger semantic matches. Relying too much on frames degraded retrieval, especially in lecture-style videos where visual information is sparse.

3. **Redundancy Removal Must Be Balanced:**
Over-aggressive filtering for redundancy (both in frames and subtitles) resulted in loss of critical semantic content, weakening the model's ability to maintain temporal or logical flow across a video.

4. **Large Language Models (LLMs) Need Structured Prompts for Grounded Responses:**
Unguided prompting often caused Gemini to hallucinate information beyond the provided video chunks. Strict, context-only prompting significantly improved factual consistency and timestamp grounding.

5. **Semantic Coherence vs Temporal Coherence Trade-off:**
Topic modeling improved semantic segmentation but often disrupted the temporal continuity necessary for retrieving usable video clips. In retrieval from sequential media like lectures, **temporal boundaries** are as important as **semantic ones**.

6. **Fixed-Window Chunking, Tuned Properly, Outperforms Adaptive Methods:**
Despite exploring dynamic chunking strategies, fixed 30-second windows produced the most reliable retrieval performance. A controlled window size ensured sufficient subtitle density and aligned better with transformer context requirements.

7. **Fine-tuning K in KNN is Crucial for Response Precision:**
Optimal retrieval performance depended heavily on selecting the right K in the nearest neighbor search. K=15 provided a good balance between diversity and relevance, avoiding both overfitting to a few chunks and diluting the context across unrelated segments.

8. **Storage Format and Preprocessing Efficiency is Crucial for Scale:**
Storing chunks in MongoDB with precomputed base64 frames allowed fast pipeline iteration. Without this, I/O bottlenecks (reading large video files repeatedly) would have made retrieval and featurization extremely slow.

Experiments to Try in the Future

1. **Enhancing Qdrant with Separate Frame Embeddings:**
   In addition to storing combined text and frame multimodal embeddings, we can insert {frame-only embeddings + metadata}, after removing redundant frames using image hashing. This would strengthen the system's ability to retrieve context based on purely visual cues when subtitle information is sparse.

2. **Incorporating Neighboring Chunks During Retrieval:**
   Extending retrieval to include the previous and next vectors (relative to the top-K nearest neighbors) would provide broader temporal context. Feeding these neighboring chunks into the LLM can improve reasoning continuity and help generate more complete answers across topic transitions.

3. **Integrating Reasoning-Optimized Language Models:**
   Deploying specialized reasoning models (such as Gemini-Pro or Claude-Sonnet) could significantly improve multi-hop inference over retrieved video segments. This would allow the system to synthesize information from multiple clips more accurately, leading to higher quality and more logically coherent responses. It would act as a validator/feedback loop where the LLM checks and makes sense of the K nearest vectors returned from Qdrant, this can improve our answers even further.

4. **Dynamic Chunk Size Based on Subtitle Density:**
   Instead of using a fixed 30-second window, future iterations could dynamically adjust chunk size based on subtitle density (e.g., words per second). This would ensure that each chunk carries a consistent amount of semantic information, improving embedding quality and downstream retrieval robustness.