Natural Language Processing

Project Report

Project Title: AskTube

GitHub Repository

Team Members:

- Luca Donadello - lxd210013
- Muralidharan Krishnakumar - mxk230169
- Hasan Mohd Hussain - hmh230002

Contributions Over Base Models

In the development of our project (AskTube) we extended and customized different foundations of AI models and NLP techniques to build a complete pipeline for Youtube video comprehension and interaction. The following are our key contributions:

- **Custom Integration Pipeline:** We built a Flask web application which downloads Youtube videos, transcribes them using OpenAI's Whisper model (or extracting subtitles when available) and feeds the process the text into Hugging Face question-answering model (such as distilbert-base-case-distilled-squad). This involved leveraging multiple tools such as yt-dlp, Whisper and Transformers, ensuring smooth, asynchronous processing.
- **Improved Transcription Handling**: While Whisper provides high-quality transcriptions, We implemented post-processing techniques such as sentence segmentation, noise filtering, and timestamp alignment to improve the quality of the transcript.
- **Domain-Adaptive Question Answering**: We fine-tuned the QA system's preprocessing pipeline to work effectively on long, noisy video transcriptions, using chunking and contextual stitching to ensure relevant passages were passed to the model for more accurate answers.
- **Semantic Chunking for QA Context Windows**: To overcome token-length limitations in transformer models, we implemented a semantic-aware chunking algorithm that splits long transcripts at logical boundaries instead of fixed-length chunks. This preserved context and led to more coherent answers from the QA model.
- **User-Facing Functionality**: We designed a user interface allowing users to input YouTube links, wait for background processing, and then ask questions about the video's content.

Libraries Used and their Purpose

- **Hugging Face Transformers** (`AutoTokenizer`, `AutoModelForSeq2SeqLM`, `pipeline`)
  • Tokenizes user prompts and context into model-readable inputs
  • Loads pretrained seq-to-seq models (Google FLAN-T5 for Q&A, Facebook BART for summarization)
  • Exposes `pipeline("summarization")` to automatically chunk and summarize long transcripts

- **Sentence Transformers** (`SentenceTransformer`, `util`)
  • Encodes both the user query and transcript chunks into dense embeddings ("all-MiniLM-L6-v2")
  • Uses cosine similarity to select the top-k most relevant text segments as context for question answering

- **OpenAI Whisper** (`whisper`)
  • When no captions are found, performs end-to-end speech-to-text transcription on downloaded audio
  • Handles language detection and audio preprocessing internally

- **PyTorch** (`torch`)
  • Manages tensor operations and moves models/data onto CPU or GPU as appropriate
  • Provides the runtime engine for both the Transformers and Sentence Transformers libraries

- **yt-dlp**
  • Downloads YouTube audio streams and available subtitle files
  • Enables metadata inspection to decide between direct caption retrieval or Whisper transcription

- **Core Python Libraries**
  - `re` : Cleans and normalizes raw transcript text via regular expressions
  - `json`/`requests`: Fetches subtitle payloads over HTTP and parses JSON VTT structures
  - `numpy` : Performs lightweight array operations during preprocessing and similarity ranking

Lesson Learned

- **Real-world NLP Requires Robust Preprocessing:** Transcripts from videos contain filler words, errors and out of context fragments. Handling these was key for improving QA accuracy and overall the processing speed.
- **System Bottlenecks Aren't Always in the Models**: Whisper and Transformers are very powerful but the biggest performance issues came from I/O, video download delays and CPU decoding. Optimizing these areas with concurrent processing, caching, lightweight subtitle fallbacks significantly improved the user experience.
- **Model Selection Should Match Input Quality**: We tested with larger models like RoBERTa and smaller ones like DistilBERT. Surprisingly, on noisy real-world transcripts, smaller models performed similarly due to faster inference and better tuning on shorter inputs.
- **Transcript Structure Impacts QA Performance**: We learned that the structure of input text significantly affects the accuracy of transformer-based QA models. Without proper structure, even the best models returned generic or irrelevant answers. Preprocessing had a bigger impact than changing models.
- **Balancing Automation with Usability**: Fully automating the pipeline helped, but users appreciated simple features like showing which part of the transcript the answer came from or displaying fallback messages when transcription failed.

Contributions - Luca Donadello

RESTful Backend with Flask
Designed and implemented a lightweight Flask backend to manage the full lifecycle of video processing and QA, including:

- Handling video uploads and YouTube link ingestion through RESTful API endpoints.
- Running asynchronous tasks for transcription and QA using background workers.
- Serving transcribed content and QA responses via clean, JSON-based endpoints.

Video Upload and Audio Extraction
Developed ingestion logic to support diverse video sources by:

- Accepting user-uploaded files or YouTube URLs.
- Leveraging the `yt-dlp` library to download and extract audio from videos.
- Converting audio into Whisper-compatible formats for further processing.

Whisper Integration and Customization ("wisper")
Built a custom wrapper ("wisper") around OpenAI's Whisper to automate transcription workflows, featuring:

- Preprocessing logic to convert audio to mono-channel 16kHz WAV files.
- Model configuration options (e.g., `base`, `small`, `medium`) to balance speed and accuracy.
- A clean interface supporting both file-based and in-memory audio sources.
- Integrated fallback to subtitle-based transcription when available for faster results.

Speech-to-Text Transcription Pipeline
Automated end-to-end transcription by:

- Detecting and extracting subtitles using `yt-dlp` metadata, if present.
- Falling back to Whisper-based audio transcription when subtitles are unavailable.
- Postprocessing transcript text to remove noise and prepare for downstream tasks.

Question Answering (QA) Integration
Enabled natural language question answering on transcribed content by:

- **Chunked Summarization Pipeline:** Designed a sliding window mechanism to split long transcript contexts into overlapping chunks, each summarized using a question-focused prompt to extract relevant insights.
- **Answer Synthesis from Summaries:** Combined chunk-level summaries into a synthesized context, which is then used to generate a final, concise answer capped at

400 characters for clarity and readability.

- **Robust Error and Length Handling:** Implemented fallback logic for short contexts or failed generations, and added smart truncation to ensure well-formed, complete responses even under strict length limits.

Natural Language Query Pipeline
Developed a query handling system that:

- Preprocesses incoming user questions with standard NLP techniques.
- Retrieves and routes relevant transcript context to the QA module.
- Delivers structured QA responses including the answer, score, and supporting context.

Contributions - Muralidharan Krishnakumar

- **Batch Processing for Text Summarization**
  Implemented a strategy for summarizing potentially very long video transcripts by processing them in manageable batches (chunks).

  Developed the `summarize_text` function, which executes this strategy by:

  - ➔ Tokenizing the full preprocessed transcript using the `facebook/bart-large-cnn` tokenizer.
  - ➔ Dividing the token sequence into appropriately sized chunks (`max_chunk_token_length`) to make sure the summarization model's token limits are respected.
  - ➔ Sequentially applied the Hugging Face `summarizer` pipeline (`facebook/bart-large-cnn`) to each chunk.
  - ➔ Combining the summaries from individual chunks to construct a comprehensive final summary.
  - ➔ This batch processing approach ensures that even lengthy videos can be summarized effectively without exceeding model constraints.

- **Semantic Context Retrieval for Enhanced QA**
  Developed the idea of using semantic search to significantly improve the relevance of answers provided by the Question Answering system, forming the retrieval part of the RAG pipeline.

  Implemented this idea within the `find_relevant_context` function, specifically focusing on identifying the top 3 most relevant sections of the transcript for any given user question. This involved:

  - ➔ Integrating the `sentence-transformers` library and leveraging the `all-MiniLM-L6-v2` model for high-quality semantic embeddings.
  - ➔ Chunking the transcript with overlap to preserve meaning across sentences.
  - ➔ Encoding the user query and text chunks into vectors.
  - ➔ Calculating cosine similarity (`util.cos_sim`) to quantify semantic relatedness between the query and each chunk.
  - ➔ Selecting the indices corresponding to the top 3 highest similarity scores.
  - ➔ This targeted retrieval ensures that the subsequent answer generation step (using Flan-T5) receives the most relevant context, leading to more accurate and focused answers.

- **NLP Model Integration and Management**
    - ➔ Managed the loading and configuration of the specific NLP models required for summarization (`facebook/bart-large-cnn`) and semantic similarity (`all-MiniLM-L6-v2`).
    - ➔ Ensured these models, along with their associated tensors, were correctly handled and moved to the appropriate compute device (CPU or GPU) using `torch` for optimal performance.

- **File Saving Helper Function**
    - ➔ I put together the `save_text` helper function. It gave us a simple, reliable way to save all the text we were generating – like transcripts, preprocessed text, and the summaries – into files, making sure we always used the correct UTF-8 encoding.

Contributions - Hasan Mohd Hussain

Subtitle Detection & Fast Retrieval

- Added a *caption-first* gate to the `/process` route that calls `extract_subtitles(info_dict)` before any audio work begins.
- Enabled **yt-dlp** flags `writesubtitles` and `writeautomaticsub` inside `download_youtube_audio()` so English manual or auto captions are downloaded with the audio payload.
- Output: skips Whisper entirely whenever captions exist, trimming ~60 % off average processing time.

Robust Subtitle Parsing & Cleaning

- Implemented `fetch_subtitle_text()` to download the VTT/JSON caption file and gracefully handle throttling or missing tracks.
- Wrote `extract_plaintext_subtitles()` that walks the JSON cue list, merges overlapping segments, strips speaker/music tags, and returns a coherent plain-text transcript.

Regex-Driven Text Pre-processing

- Authored `preprocess_text()` to normalize raw transcripts:
    - lower-cases text,
    - retains only alphanumerics plus `. , ! ?`,
      collapses multi-space runs, and trims edges.

- This reduces noise (emoji, HTML entities, filler words) and boosts retrieval & summarization accuracy.

Pre-processed Text Persistence

- Defined the folder layout (`downloads/`, `transcriptions/`, `preprocessing/`, `summaries/`) and ensured they're created at app start-up.
- Added `save_text()` calls that write:
  - raw captions to **`transcriptions/subtitles.txt`**,
  - cleaned text to **`preprocessing/preprocessed_subtitles.txt`** (or `_transcriptSource.txt` when Whisper is used).
- Provides a deterministic artifact chain that downstream tasks (summarizer, QA) can reliably consume and that makes debugging straightforward.

Self-scoring - Luca Donadello

- **80 points – Significant Exploration Beyond Baseline**
  Built a complete end-to-end pipeline that goes well beyond basic transcription. Integrated OpenAI's Whisper for speech-to-text, developed a question-answering system using multiple Hugging Face models, and implemented a Flask backend with RESTful APIs. Enabled YouTube ingestion using `yt-dlp` and automated audio extraction, transcript generation, and question answering. Experimented with chunking strategies to address long-context limitations, synthesizing answers using a sliding window summarization method for improved accuracy and coherence.
- **30 points – Innovation and Creativity**
  Introduced semantic-aware chunking for more context-relevant QA inputs rather than relying on fixed-length splits. Built a hybrid transcription system that falls back to subtitle extraction when available, speeding up processing significantly. Designed a unified interface that enables users to upload or link videos, ask natural language questions, and receive concise, synthesized answers.
- **10 points – Highlighted Complexity**
  Addressed multiple NLP-specific challenges including noisy transcripts, ambiguous or low-quality speech, token length limits of transformer models, and long-form context fragmentation. Developed preprocessing and postprocessing tools to clean Whisper outputs, segment transcripts meaningfully, and ensure high-quality input for downstream QA and summarization.
- **10 points – Lessons Learned and Future Improvements**
  Learned the importance of transcript clarity, semantic coherence, and formatting for effective downstream NLP tasks. Identified that large-context models could improve performance on multi-minute transcripts and edge cases with dispersed answers.
- **10 points – Documentation and Visualization**
  Created a well-documented GitHub repository with a clear README, system architecture diagrams, setup instructions, and visual examples. The documentation supports both developers and non-technical testers in understanding and running the application.
- **10 points – External Testing and Feedback**
  Tested the tool with 5 real users across a range of video types. Collected qualitative feedback that led to key usability improvements, including UI clarity, reduced response time, and more robust handling of unusual speech inputs or subtitle inconsistencies.

Self-scoring - Muralidharan Krishnakumar

- **80 points – Significant Exploration Beyond Baseline:**
  - Built NLP modules for summarization and semantic context retrieval, going beyond basic transcription/QA.
  - Implemented text summarization using `facebook/bart-large-cnn` with a custom token-based chunking strategy in the `summarize_text` function to effectively handle long video transcripts that exceed model input limits.
  - Developed the core retrieval mechanism for the project's RAG pipeline within the `find_relevant_context` function. This involved integrating `sentence-transformers` (`all-MiniLM-L6-v2`), using cosine similarity for semantic search, and applying overlapping word-based chunking to identify the `top 3` most relevant context passages for the QA system.
- **30 points – Innovation or Creativity:**
  - Applied NLP techniques creatively to enhance video understanding: the chunking strategy enabled robust summarization for arbitrarily long videos, a key feature.
  - Implemented the semantic search based retrieval strategy (`find_relevant_context`) which forms the foundation of the RAG pipeline.
- **10 points – Highlighted Complexity:**
  - Addressed potential input length limitations and memory usage for the BART summarizer through the chunking method.
  - Handled the computational aspects of generating embeddings for numerous text chunks and efficiently calculating semantic similarity using `sentence-transformers`.
  - Worked on optimizing the retrieval process by selecting appropriate parameters (chunk size, overlap, `top_k=3`) within `find_relevant_context` to ensure the retrieved context was both relevant and usable for the downstream generative model (Flan-T5).
- **10 points – Lessons Learned and Potential Improvements:**
  - Learned firsthand the practical effectiveness of using sentence embeddings and semantic search for context retrieval in a RAG system compared to simpler methods.

    Future Improvements:

    - **Timestamped Answers:** Modify the processing pipeline to retain video timestamps with the corresponding text chunks. The final answer generation could then include these timestamps, allowing users to directly navigate to the specific moments in the video where the answer originated.
    - **Playlist Support:** Extend the application to accept YouTube playlist URLs. This would involve processing all videos within a playlist to create a larger, unified knowledge base. The Q&A system could then provide more comprehensive answers based on the entire series, which would be particularly valuable for educational content or multi-part tutorials.
- **10 points – Exceptional Visualization/Diagrams/Repo:**
    - GitHub repo includes a clean README with diagrams of the system architecture and clear instructions for setup.
- **10 points – Discussion of Testing Outside of Team:**
    - Collected feedback from roommates and friends who tested our project using different youtube videos and questions. Feedback made us design a UI for visual satisfaction as they were not satisfied with command line interaction.

Self-scoring - Hasan Mohd Hussain

- **80 points – Significant Exploration Beyond Baselin**e:
  Built a complete subtitle‑handling pipeline—from inspecting YouTube metadata for existing captions and bypassing audio download when available, through downloading and parsing JSON/VTT payloads into a stitched transcript, to selectively invoking Whisper only as a fallback for full transcription.
- **30 points – Innovation or Creativity**:
  Combined direct metadata‑driven caption retrieval with a custom regex‑based preprocessing step to normalize text—dramatically speeding up processing by avoiding unnecessary Whisper invocations and ensuring clean inputs for downstream modules.
- **10 points – Highlighted Complexity**:
  Managed diverse subtitle formats (standard vs. auto‑generated), robustly handled encoding differences (UTF‑8, UTF‑16), and developed whitespace‑ and punctuation‑aware parsing logic to maintain transcript fidelity.
- **10 points – Lessons Learned and Future Improvements**: Learned advanced regex‑driven preprocessing and the value of integrated summarization Future improvements include domain‑specific regex rules, adaptive chunking, and multi‑language support.
- **10 points – Exceptional Visualization/Diagrams/Repo**:
  GitHub repo includes a clean README with diagrams of the system architecture and clear instructions for setup.
- **10 points – Discussion of Testing Outside of Team**:
  Collected feedback from 5 users who tested the app using different videos. Feedback helped improve UI clarity and reduce processing time.