Name: Abhiram Koppuravuri

Mail: abhiramkoppuravuri@gmail.com

Phone: 6304760789

Github: https://github.com/Abhiram-koppuravuri/MultimodalRAG

Multimodal RAG System: End-to-End Workflow

Our Multimodal Retrieval-Augmented Generation (RAG) system enables efficient retrieval and generation of insights from diverse data modalities such as video, audio, text, and images. Below, we provide a comprehensive breakdown of the workflow:

1. Data Processing

The system begins by converting raw, heterogeneous input data into normalized textual representations suitable for downstream processing.

Input Sources:

• Videos: YouTube videos are downloaded and converted to audio files in .mp3 format.

• PDF Documents: Text and images are extracted directly from PDFs.

Processing Steps:

• Video Transcription:
- Audio from .mp3 files is transcribed into text using the Whisper model, which is robust for multilingual and noisy audio data.

• Text Extraction from PDFs:
- Raw text is extracted, while images within the PDF are identified and isolated for further processing.

• Image Captioning:
- Captions summarizing extracted images are generated using the Salesforce/blip-image-captioning-base model, which provides a concise textual description of image content.

• Summarization:
- Lengthy textual content, whether transcriptions or extracted text, is summarized to retain key insights and improve query retrieval efficiency.

Output: The processed data is stored in a unified textual format, creating a consistent foundation for embedding and retrieval.

2. Embedding

The normalized textual data is transformed into a vector representation using a pre-trained embedding model. This step ensures data from different modalities (audio, text, image) is mapped into a shared semantic space.

• Model: We use sentence-transformers/all-MiniLM-L6-v2, a lightweight and efficient model for sentence-level embeddings.

• Embedding Space: This model generates fixed-length vector representations, enabling the comparison and retrieval of semantically similar content across various data types.

3. Retriever

The retriever component identifies the most relevant pieces of information from the pre-embedded dataset in response to a user query.

• Query Embedding:
- User queries are transformed into vector representations using the same embedding model (sentence-transformers/all-MiniLM-L6-v2) to maintain consistency.

• Nearest Neighbor Search:
- The system performs a nearest neighbor search in the vector database (e.g., FAISS, Pinecone) to identify the top-N relevant chunks based on vector similarity.

4. Augmented Generation

This step combines the retrieved chunks with the user query to create an augmented input for the generative model.

• Augmentation:
- Retrieved chunks (e.g., text passages, image captions) are concatenated with the user's original query to provide context and improve the relevance of generated answers.

• Generative Model:
- The augmented input is fed into the facebook/bart-base generative model, which produces a coherent and contextually rich response.

• Output:
- The generated response is designed to address the query comprehensively, incorporating insights from all relevant data modalities.

5. User Interface

The final step ensures the system is accessible and intuitive for end users.

• Implementation: The interface is built using Streamlit, a Python-based framework for rapid development of interactive web applications.

• Features:
  - Query input box for user submissions.
  - Real-time display of generated responses.
  - Logs or visualization components (optional) for debugging and monitoring.

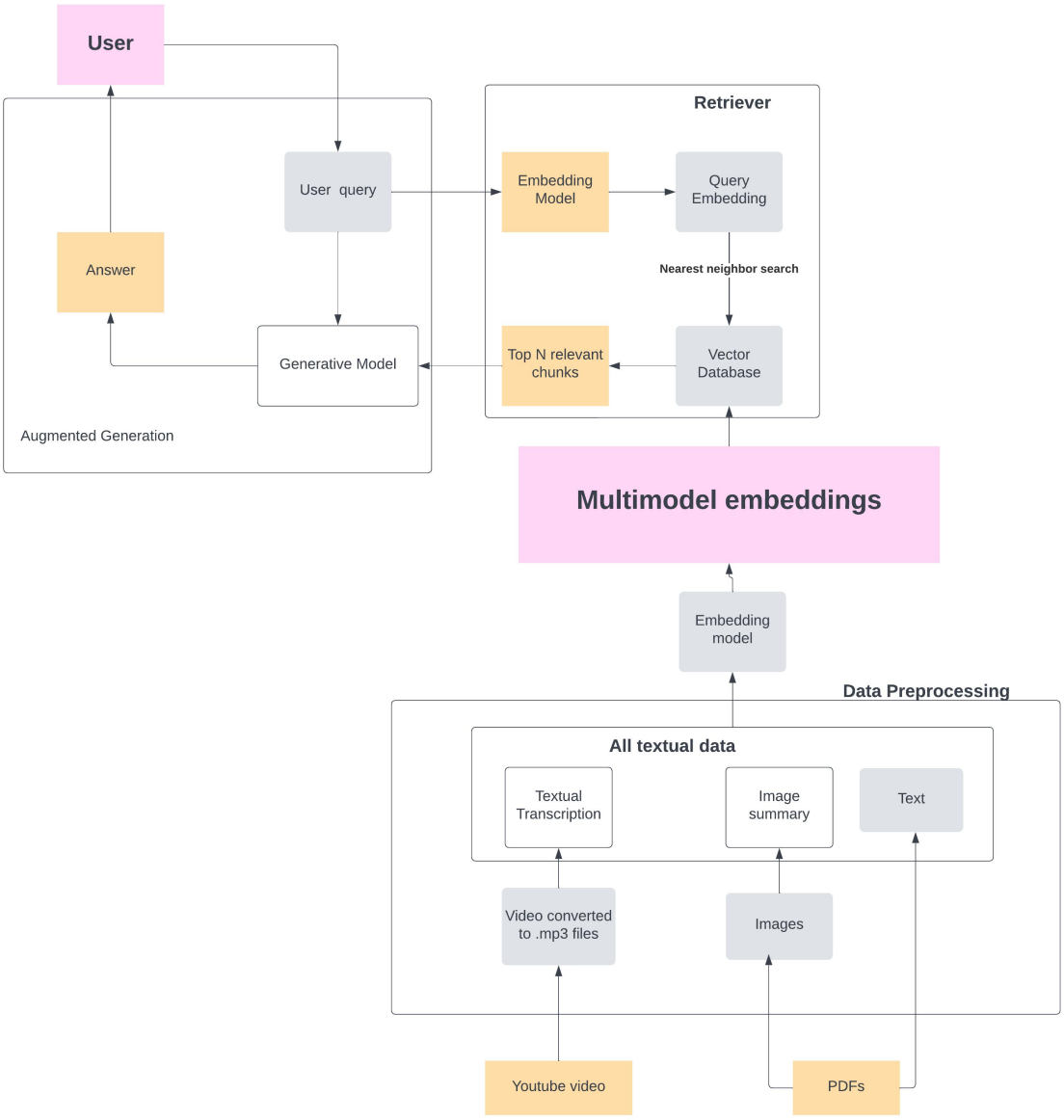
System Highlights

• Multimodal Support:
  - Unified embedding space enables seamless integration and comparison of audio, text, and image data.

• Scalability:
  - Modular design allows for easy expansion to support additional data types and larger datasets.

• Efficiency:
  - Combining retrieval and generative approaches reduces computational overhead while ensuring high-quality results.

• Accessibility:
  - The user-friendly Streamlit interface simplifies interactions, making the system accessible to non-technical users.



Detailed overview of components

Video

1. YouTube Audio Downloader

This component downloads audio files from YouTube videos and saves them as `.mp3` files in a specified folder.

Key Features:

- Converts YouTube video URLs to audio files using **yt-dlp**.
- Ensures filenames are safe and compatible with file systems.
- Supports batch downloading from multiple video URLs.
- Tracks downloaded files with a dictionary linking URLs to file paths.

Classes and Methods:

- `YouTubeAudioDownloader`:
  - `get_safe_filename`: Sanitizes filenames to remove invalid characters.
  - `download_audio`: Downloads a single video's audio and converts it to `.mp3`.
  - `download_multiple_audios`: Handles multiple URLs, downloading audio for each.

---

2. Audio Transcriber

This component uses OpenAI's **Whisper** model to transcribe the downloaded audio files into text.

Key Features:

- Transcribes `.mp3` files using the **medium** variant of the Whisper model.
- Automatically detects empty or invalid audio files to avoid processing errors.
- Provides a dictionary of transcriptions linked to their respective video URLs and file paths.

Classes and Methods:

- `AudioTranscriber`:
  - `transcribe_audio`: Processes a single audio file and returns its transcription.
  - `transcribe_all_audios`: Handles batch transcription for multiple audio files.

---

3. Workflow

- **Initialization**:
    - Specify the output folder for downloaded audio files.
    - Load the **Whisper model** (`medium` size) on a GPU (`cuda`) or CPU as fallback.
- **Execution**:
    - **Step 1**: Download audio files from a list of YouTube URLs using the `YouTubeAudioDownloader`.
    - **Step 2**: Transcribe the downloaded audio files using the `AudioTranscriber`.
    - **Step 3**: Organize transcriptions into a structured format for further use.

---

4. Output

The system generates:

1. **Audio Files**: Saved as `.mp3` files in the specified folder.
2. **Transcriptions**: Stored in a dictionary format, where each entry contains:
    - `url`: The YouTube video URL.
    - `audio_path`: Path to the downloaded `.mp3` file.
    - `transcription`: Transcribed text from the audio file.
3. **JSON Summary**: The transcription results are serialized into a JSON object for easy sharing or integration.

2. PDF

1. PDF Processing

The report is processed using the `unstructured` module, which extracts both text and image data from the PDF.

Dependencies:

- **Tesseract OCR**: Used for high-resolution image extraction from the PDF.
- **Poppler**: Backend for handling PDF rendering and image extraction.

Key Features:

- Extracts both textual and visual components from the PDF.
- Saves images to a specified directory while retaining metadata such as page numbers.

---

2. Textual Component Extraction

Functionality:

- Extracts **narrative text** from the PDF.
- Metadata includes:
  - Source document path.
  - Page number.
  - Paragraph number.
  - Text content.

Implementation:

- `extract_text_with_metadata`: Processes the extracted data and organizes it into a structured format for downstream analysis.

Output:

- A list of dictionaries, where each dictionary represents a paragraph of text with its metadata.

---

3. Visual Component Extraction

This module extracts images from the PDF and generates metadata, such as page numbers and accompanying text.

3.1 Image Metadata Extraction

Functionality:

- Extracts images along with their metadata (e.g., page number and any adjacent textual descriptions).
- Handles missing or unavailable image paths gracefully.

Functions:

- `extract_image_metadata_with_text`: Links images to their context, capturing any text immediately following the image.

3.2 Displaying Images

Functionality:

- Displays images alongside their associated metadata using **matplotlib**.
- Annotates images with text below them for better visualization.

Functions:

- `display_images_with_text_from_metadata`: Visualizes images with contextual annotations.

---

4. Image Summarization

Model: Hugging Face's **BLIP (Bootstrapped Language-Image Pretraining)** for image captioning.

Processor: `BlipProcessor`, which prepares images for the model.

Functionality:

- Generates textual summaries for extracted images.
- Encodes image files as Base64 for easy storage or transmission.

Functions:

- `extract_image_metadata_with_summary`: Combines metadata extraction with AI-based image captioning.

Output:

- A list of dictionaries containing:
  - Page number.
  - Image path.
  - Generated description (image summary).
  - Base64-encoded image.

Vector Database

- **Weaviate Connection and Schema Setup**:
    - • Establishes a connection to the Weaviate WCS instance using API key authentication.
    - Defines the schema for the `RAGESGDocuments` collection with properties like `source_document`, `text`, `audio_path`, `description`, etc.
- **Data Ingestion**:
    - • Functions (`ingest_audio_data`, `ingest_text_data`, `ingest_image_data`) handle data ingestion for various modalities.
    - The use of dynamic batching optimizes data uploads to the Weaviate instance.
- **Embedding Generation**:
    - • Utilizes Hugging Face's `sentence-transformers/all-MiniLM-L6-v2` model to generate embeddings for vector-based searches.
- **Multimodal Search**:
    - • Implements a vector-based search using Weaviate's `near_vector` query to retrieve documents based on user queries.
- **Result Formatting**:
    - • Nicely formats and prints the search results, distinguishing between text, audio, and image data.
- **Interface**:
- **Core Functionalities**:
    - • **Data Retrieval**: Uses the `search_multimodal` function to query the Weaviate instance.
    - **Context Building**: Constructs a detailed context from retrieved documents for response generation.
    - **Response Generation**: Employs the `facebook/bart-base` model to generate context-aware answers.

- **Streamlit Integration**:
    - Provides a user-friendly interface to input queries and display AI-generated responses alongside retrieved sources.
- **Hugging Face Integration**:
    - The `sentence-transformers/all-MiniLM-L6-v2` is an excellent choice for lightweight embedding generation.

Links:

Github: https://github.com/Abhiram-koppuravuri/MultimodalRAG

Lucid chart: https://lucid.app/lucidchart/04fbde81-9a2d-4783-977f-b31ffd3234ef/edit?viewport_loc=-2272%2C-983%2C3971%2C1750%2C0_0&invitationId=inv_e0632be2-dcea-4761-bf61-a4e290435882