CourseBuddyAI

Team members:

Avinash Badeo (abaldeo2@illinois.edu)

Zach Pohl (zcpohl2@illinois.edu)

Colton Bailey (coltonb4@illinois.edu)

Kacper Dural (kdural2@illinois.edu)

CS410 Text Information Systems

Fall 2023

University of Illinois at Urbana-Champaign

Overview

CourseBuddyAI is a Chrome Extension designed enhance the Coursera Online Learning Platform using the power of ChatGPT!

🚀 **Features:**

- It gives students the ability to summarize lecture video transcripts into set of bullet points, automating note-taking.
- It also provides a chatbot interface that allows students to have their questions answered based on the uploaded course content.
- It works by using by combining Generative and Retrieval models, allowing for getting more contextually grounded responses from a large language model (LLM), which in this case is the OpenAI GPT3.5 model.

Motivation

- The main motivation is to help enhance student learning while watching lecture videos and answering questions they may have on lecture topics.
- We felt this would help reduce the burden of TAs having to answer the same question repeatedly on Campuswire and reduce the response time for students.
- We also wanted to explore using a state-of-the-art LLMs to perform NLP tasks such as text summarization and Question-Answering using information retrieval.
- Although a large language model may already be trained to answer simple questions such as “What is precision”, different courses may have different terminology or explanation for the same concepts, so it’s important for it to be able to answer the students question guided by the course context.

Technical Design

Background on Retrieval Augmentation Generation (RAG)

- Documents are loaded from a database and then split up into smaller chunks to be embedded.
- The chunks of text are passed through an embedding model that generates dense vectors that get stored in a vector database.
- When a question is asked by the user, the system takes the embedded query vector then performs similarity search to find the most relevant pieces of the content.
- The users question along with the retrieved pieces of context are passed to the LLM prompt to get a accurate and relevant response, mitigating the potential of hallucinations.

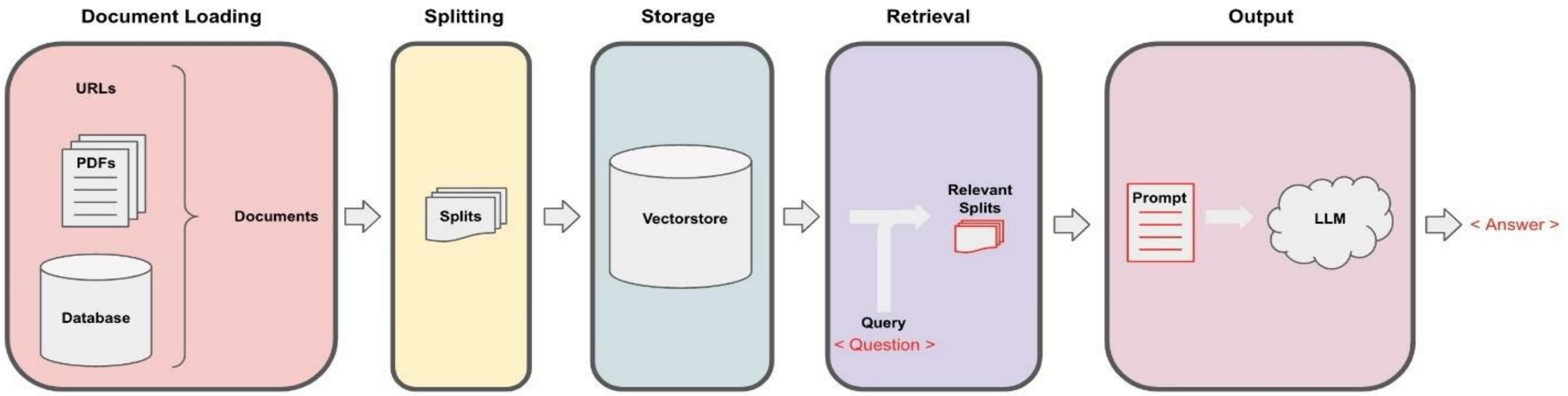


Figure 1

High-Level Architecture Diagram

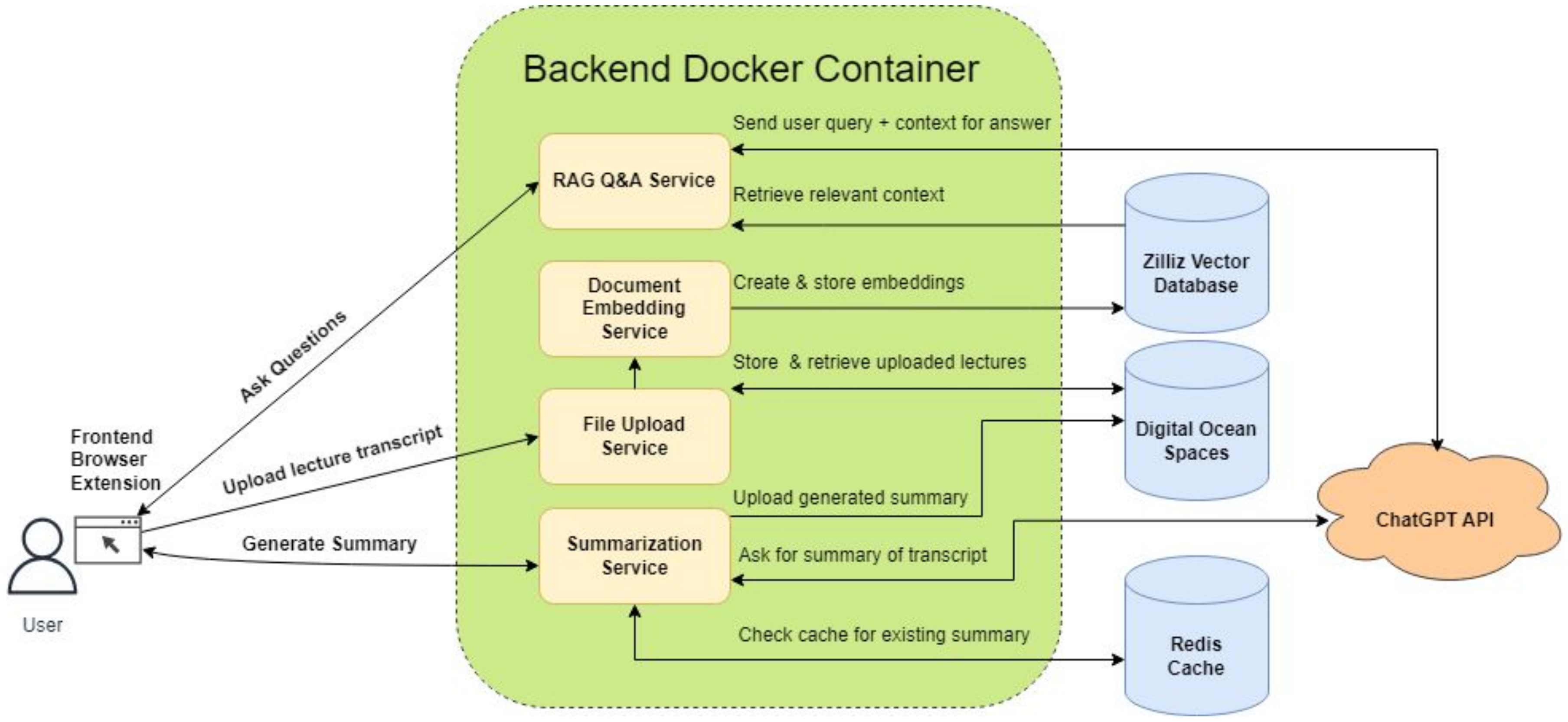

Backend Docker Container
Send user query + context for answer
RAG Q&A Service
Retrieve relevant context
Zilliz Vector Database
Document Embedding Service
Create & store embeddings
Store & retrieve uploaded lectures
Digital Ocean Spaces
File Upload Service
Ask Questions
Upload lecture transcript
Frontend Browser Extension
Upload generated summary
ChatGPT API
Generate Summary
Summarization Service
Ask for summary of transcript
User
Check cache for existing summary
Redis Cache

Figure 2

Tech Stack

Backend:

- Python 3.11
- FastAPI for backend REST Services
- Langhchain & Openai for interfacing with ChatGPT
- Unstructured for document loading
- Boto3 for uploading/retrieving documents to S3
- SqlAlchemy with Alembic for SQL database
- Redis for caching
- Pymilvus for Vector DB
- Docker & Github Codespaces for easier development

Frontend:

- React 16 with Typescript, Redux, and react-router
- Plasmo Framework for Chrome extension development
- Material-ui for styling

Infrastructure & Hosting

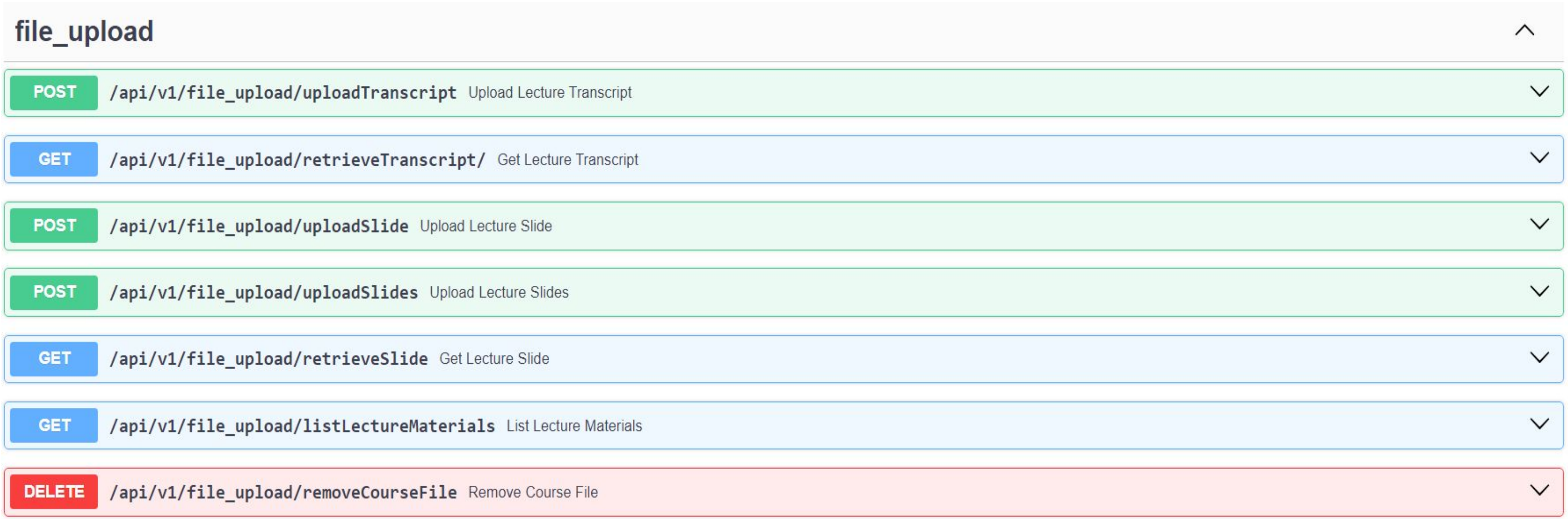
- Digital Ocean Spaces for File Hosting - https://www.digitalocean.com/products/spaces
- Zilliz Vector Database for Serverless Vector Database - https://zilliz.com/
- Upstash Redis for Serverless Redis instances - https://upstash.com/
- YugabyteDB for managed Cloud PostgreSQL Database - https://www.yugabyte.com/
- Github Actions Workflow for automated deployments of the backend to GCP
- Google Artifact Registry for storing docker container builds
- Google Cloud Run for deploying & running the containerized FastAPI backend

Backend Implementation Details

File Upload Service

- When a User wants to upload additional course slides & transcripts they can go to File Upload drop down in the extension popup and select files to add.
- The File Upload Service takes the uploaded lecture transcripts & slides then persists them to a DigitalOcean Spaces bucket compatible with the AWS S3 API.
- This allows for the retrieval of the stored documents by the Document Embedding & Summarization Services.
- It also provides a endpoint for the frontend to get a list of uploaded documents to display to the user. When clicked on the filename, it will fetch the summary generated for that document.

File Upload Endpoints



file_upload

POST /api/v1/file_upload/uploadTranscript Upload Lecture Transcript

GET /api/v1/file_upload/retrieveTranscript/ Get Lecture Transcript

POST /api/v1/file_upload/uploadSlide Upload Lecture Slide

POST /api/v1/file_upload/uploadSlides Upload Lecture Slides

GET /api/v1/file_upload/retrieveSlide Get Lecture Slide

GET /api/v1/file_upload/listLectureMaterials List Lecture Materials

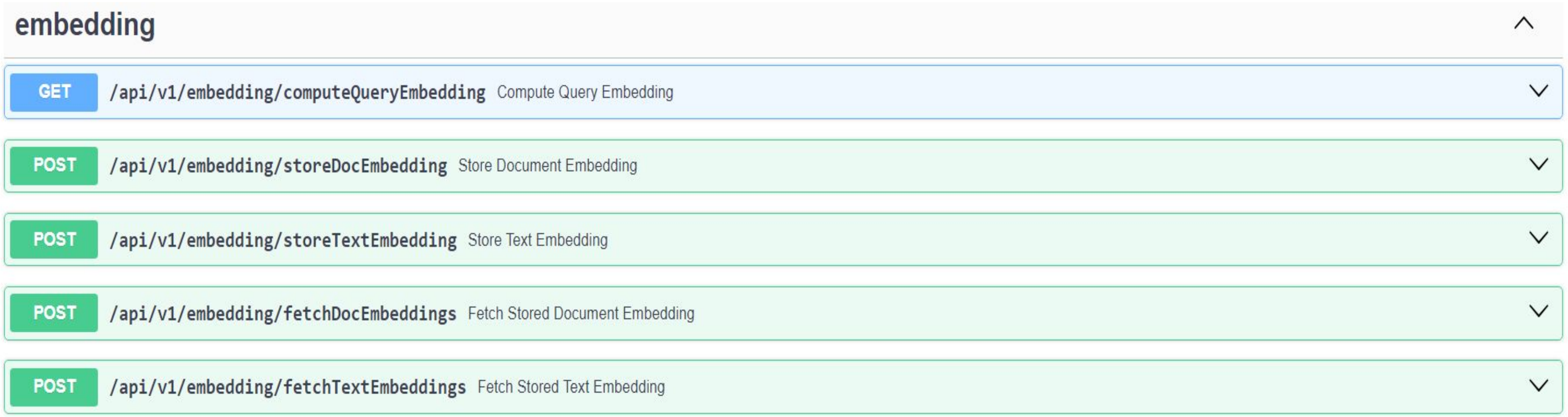
DELETE /api/v1/file_upload/removeCourseFile Remove Course File

Figure 3

Document Embedding Service

- The document embedding service is used to generate text embeddings for uploaded lecture transcripts.
- It first splits the documents into smaller chunks of 500 tokens to fit the context window.
- The HuggingFace BGE text embedding model maps the text chunks into dense-vectors (768 dimensions)
- The BAAI/bge-base-en-v1.5 model has shown state of the art performance in the MTEB leaderboard for semantic search and is small enough to be run locally.
- After the document vectors are generated, the embedding service stores and indexes in cloud-hosted Milvus vector database instance.
- This allows for Q&A Service to quickly retrieve the relevant documents for the user's question based on similarity measure (L2 distance).

Embedding Endpoints


embedding

GET /api/v1/embedding/computeQueryEmbedding Compute Query Embedding

POST /api/v1/embedding/storeDocEmbedding Store Document Embedding

POST /api/v1/embedding/storeTextEmbedding Store Text Embedding

POST /api/v1/embedding/fetchDocEmbeddings Fetch Stored Document Embedding

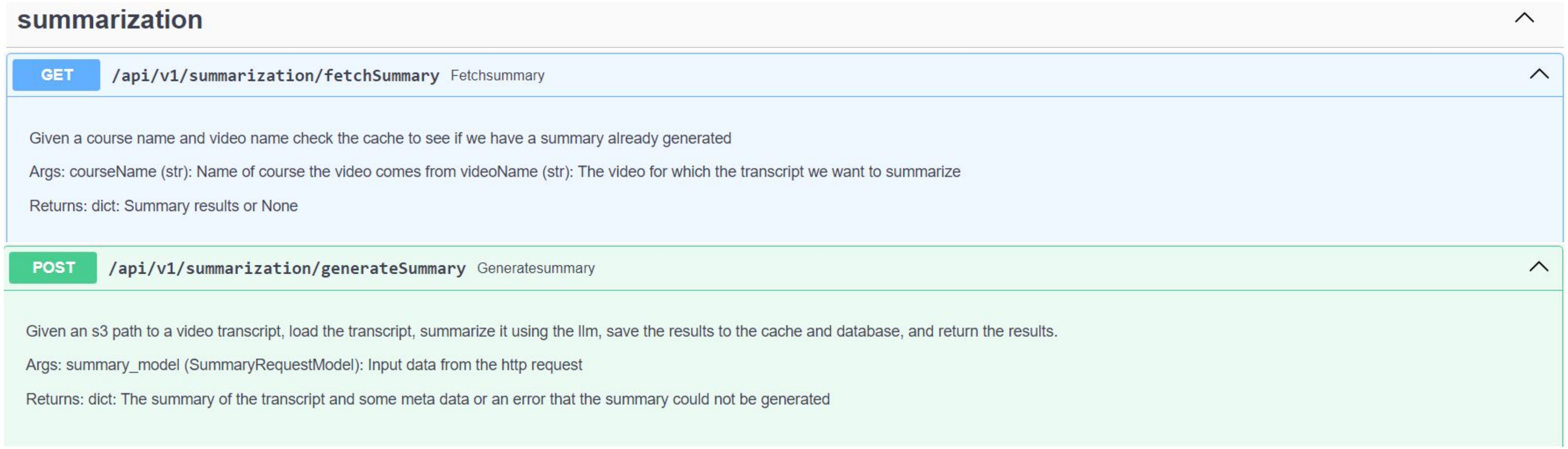
POST /api/v1/embedding/fetchTextEmbeddings Fetch Stored Text Embedding

Figure 4

Summarization Service

- When a user wants a summary of a lecture video transcript they can click on the "Generate Summary" button located under the Coursera video player.
- The summarization service takes the uploaded text and generates a bullet-point summary using the ChatGPT API.
- It dynamically calculates the number of bullet points based on the number of tokens in the document.
- If the document is longer than the LLM context window (4096), it is split up into chunks and then the summaries for each chunk is combined into one summary.
- The summarization service uploads the generated summaries to the S3 bucket. This avoids having to regenerate a summary for the same video multiple times.
- The summarization service also caches the summarizes in Redis for fast retrieval.

Summarization Endpoints



GET /api/v1/summarization/fetchSummary Fetchsummary

Given a course name and video name check the cache to see if we have a summary already generated

Args: courseName (str): Name of course the video comes from videoName (str): The video for which the transcript we want to summarize

Returns: dict: Summary results or None

POST /api/v1/summarization/generateSummary Generatesummary

Given an s3 path to a video transcript, load the transcript, summarize it using the llm, save the results to the cache and database, and return the results.

Args: summary_model (SummaryRequestModel): Input data from the http request

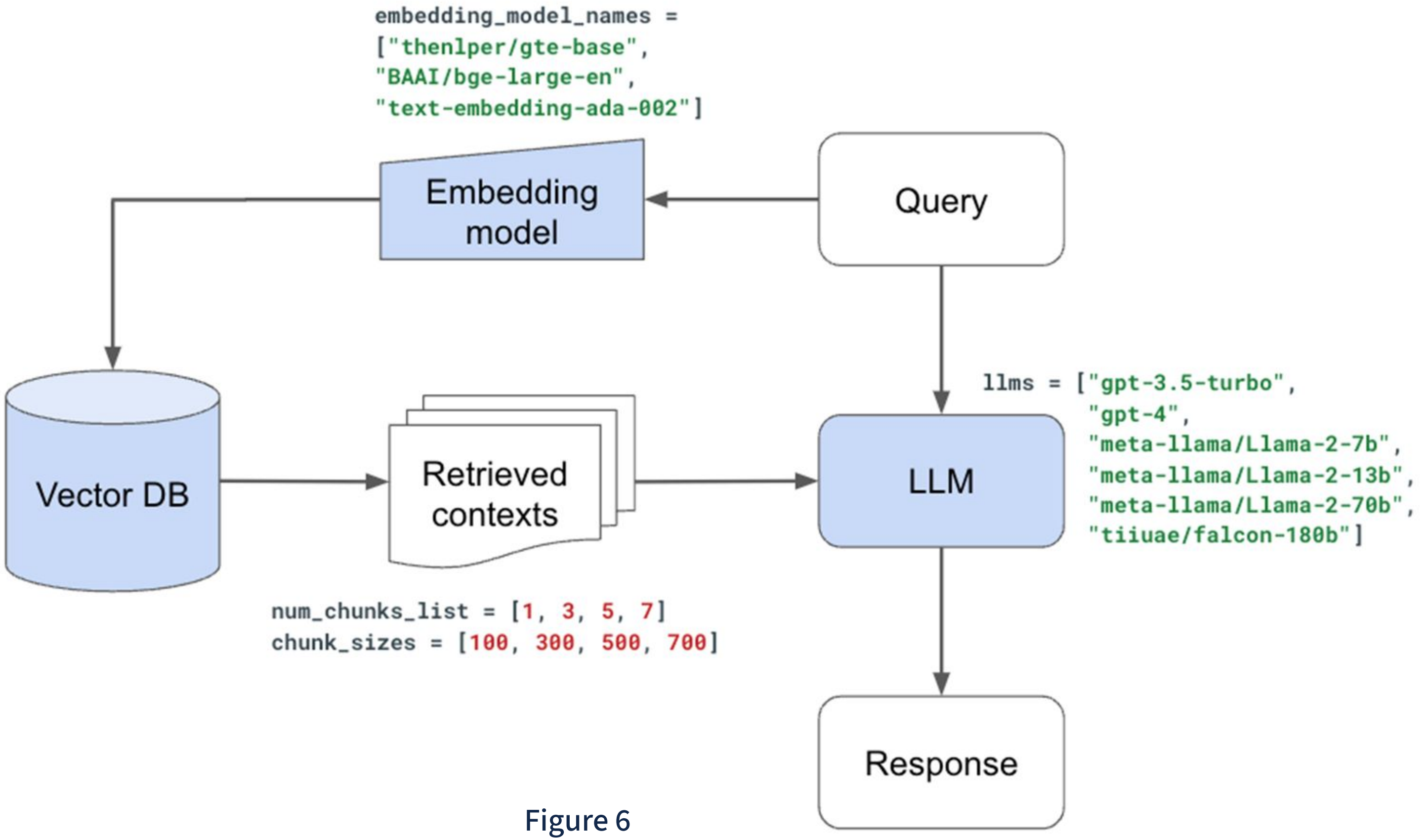
Returns: dict: The summary of the transcript and some meta data or an error that the summary could not be generated

Figure 5

RAG Q&A Service

- The RAG Service is used by our Q&A chatbot interface to provide ChatGPT with the users' questions and the relevant lecture documents in order for it to synthesize an answer that is contextually relevant and based on the course material.
- To improve system response times, we cache the responses generated by ChatGPT. This means that if the same question is asked again, the system can retrieve the answer from the in-memory cache.
- We also explored and implemented a hybrid search approach to enhance context retrieval. This approach combines the vector based semantic search with BM25 search, using the reciprocal rank fusion algorithm.
- However we found that using the hybrid search approach had worse responses and decided to stick with using similarity based search and Maximal Marginal Relevance (MMR) ranking.

RAG with Semantic Search



```
embedding_model_names =
["thenlper/gte-base",
"BAAI/bge-large-en",
"text-embedding-ada-002"]
```

```
llms = ["gpt-3.5-turbo",
  "gpt-4",
  "meta-llama/Llama-2-7b",
  "meta-llama/Llama-2-13b",
  "meta-llama/Llama-2-70b",
  "tiiuae/falcon-180b"]
```

```
num_chunks_list = [1, 3, 5, 7]
chunk_sizes = [100, 300, 500, 700]
```

Figure 6

RAG with Hybrid Search

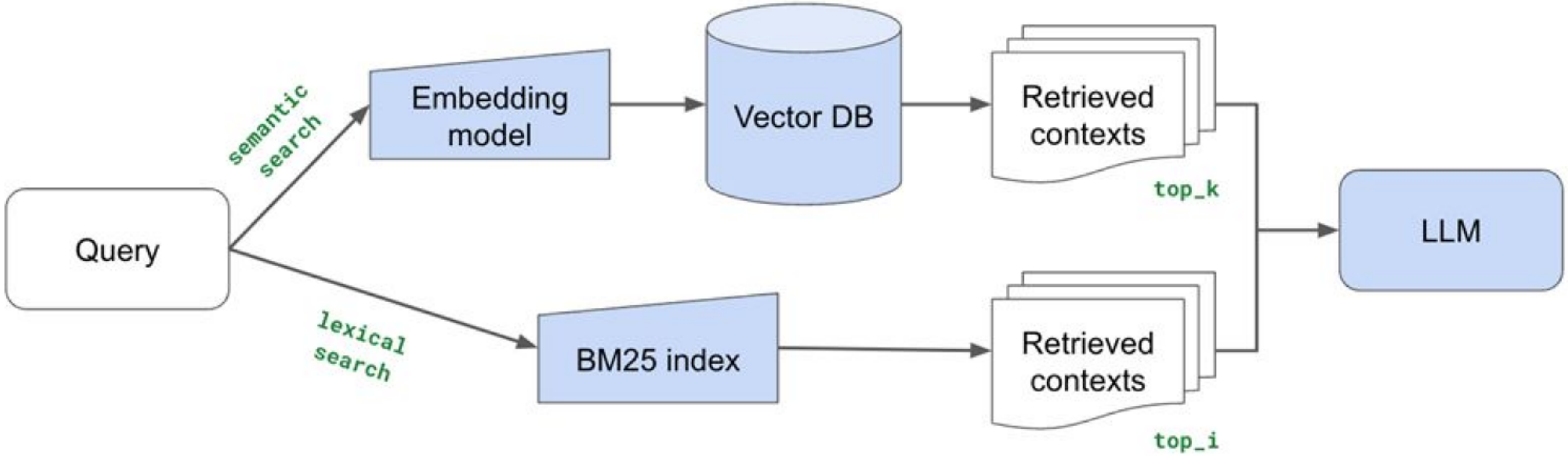
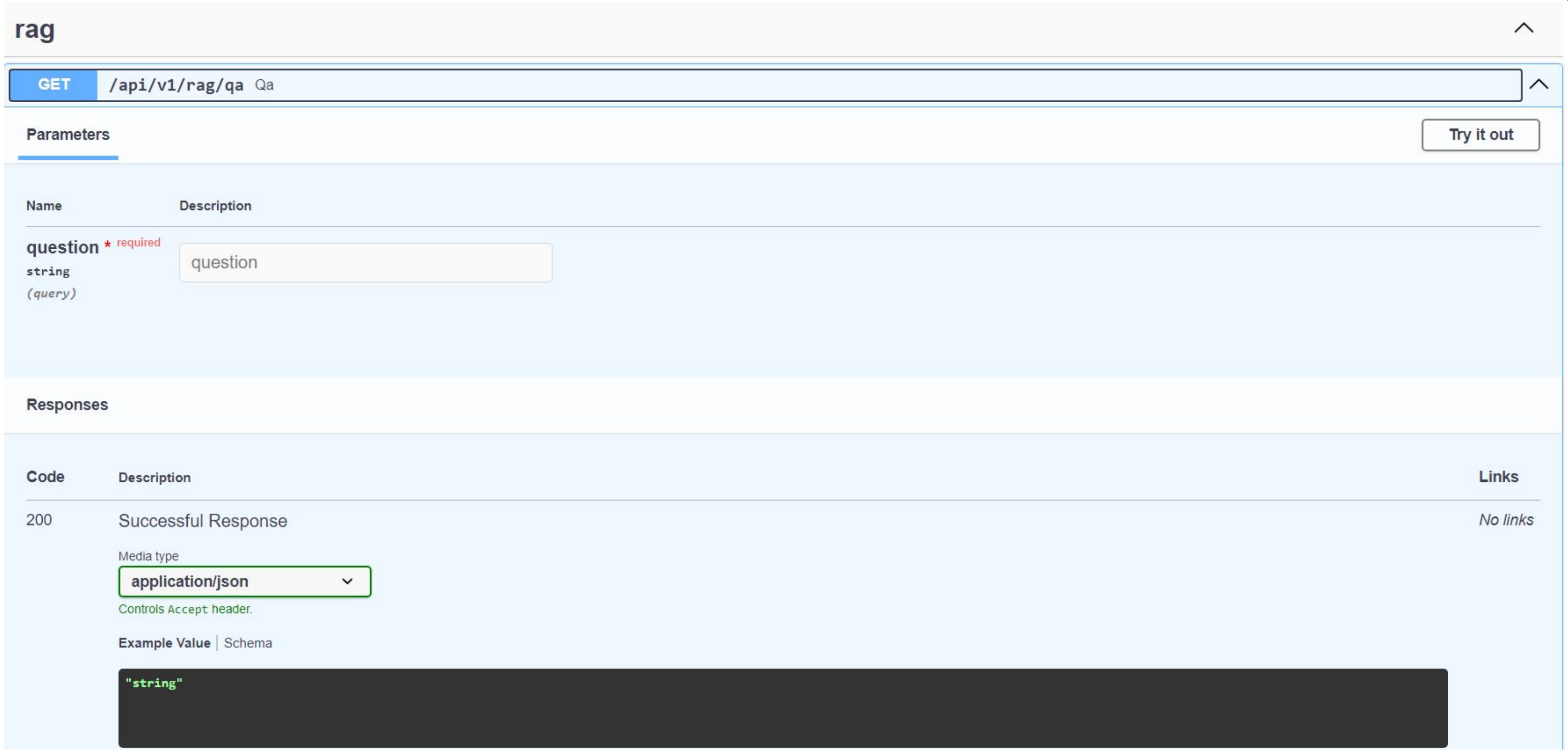


Figure 7

RAG Endpoints


rag

GET /api/v1/rag/qa Qa

Parameters

Try it out

Name	Description
question * required string *(query)*	question

Responses

Code	Description	Links
200	Successful Response Media type application/json Controls Accept header. Example Value | Schema	*No links*

```
"string"
```

Figure 8

Installation & Demo

Software Usage

How to Use:

● Install the CourseBuddyAI Chrome extension.

● Go to the Coursera website and select a course video that you are interested in.

● Click the "Generate Summary" button to get a concise summary of the course content.

● Upload the lecture transcripts using the file upload interface.

● Use the Q&A chatbot interface to ask questions and get answers based on the uploaded course content.

Refer to full user guide & installation instructions here: https://github.com/abaldeo/CS410_CourseProject/tree/main/docs

Software Demo…

Conclusion

Challenges Faced

- Evaluation Benchmark & Dataset Generation - Team member that was supposed to work on this suddenly left the team in middle of the project without making contributions.
- OpenAI API - faced service outages and rate limits caused by public ChatGPT usage.
- Chrome Extension development issues - faced issues with setting the correct permissions for the extensions and keybind conflicts with Coursera website.
- Learning Plasmo framework - Limited documentation & sometimes conflicted with Chrome default behavior.
- Automated Deployment to Google Cloud Run - faced issues with IAM permissions and Github Workflow Actions pipeline setup

Future Work

- Full Support for PDF, Powerpoint & other document types
- Voice-chat feature (TTS/STT) for Chatbot interface
- Explore Query-rewriting and expansion using ChatGPT
- Semantic Caching for Question & Answering
- Complete RAG Benchmarking & Automated Evaluation Pipeline

Thank You!