CS 410 Project Progress Report

1. Which tasks have been completed?

For the course project our team has been working on building a Chrome extension that gives Coursera students the ability to upload & summarize lecture transcripts/slides. It also allows them to have their questions answered through a chatbot interface that utilizes the Retrieval Augmented Generation (RAG) technique for getting more relevant responses from a large language model (LLM), which in this case is the OpenAI GPT3.5 model that we are using. To date we have made decent progress on the project and have completed about 65% of the planned tasks listed in our initial project proposal (See table at the end of report).

For the frontend component we are largely done with development of the chrome extension user interface which is implemented using the React and Plasmo frameworks. This includes the UI of the AI powered chatbot for answers to lecture questions, the file upload button which allows for lecture slides/pdfs to be indexed into the database, and an embedded button on the coursera lecture page that gets a summary of the current video when clicked on. We have also added a user login flow into the extension popup which provides a login/signup page and logout functionality.

Chrome Extension UI Screenshots

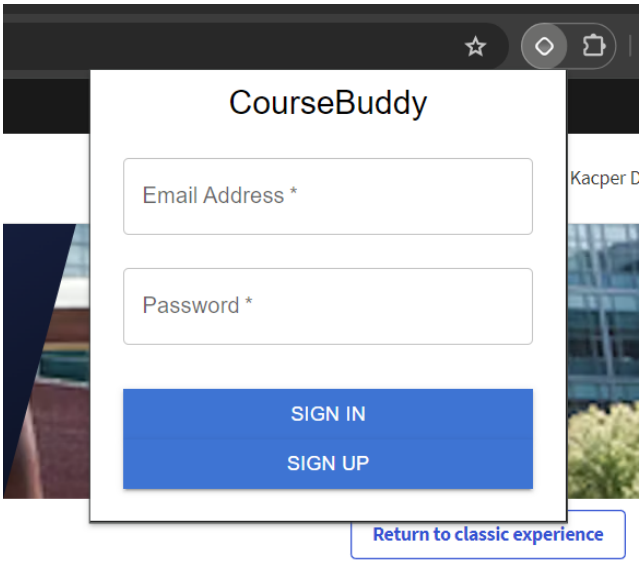

Figure 1A: Login/Signup Panel

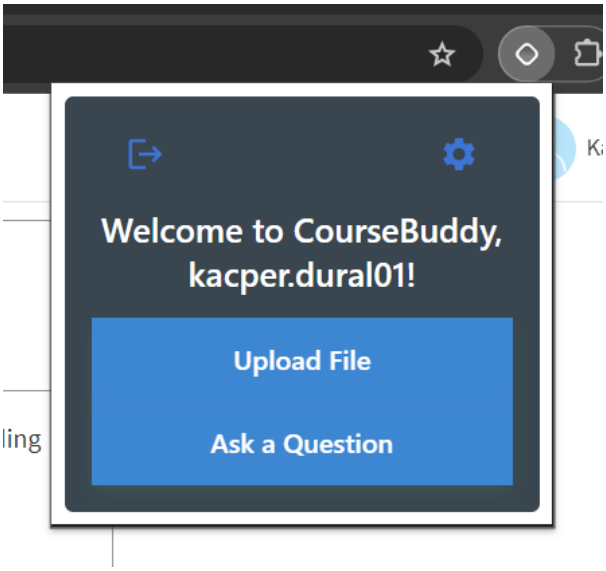

Figure 1B: Extension Popup

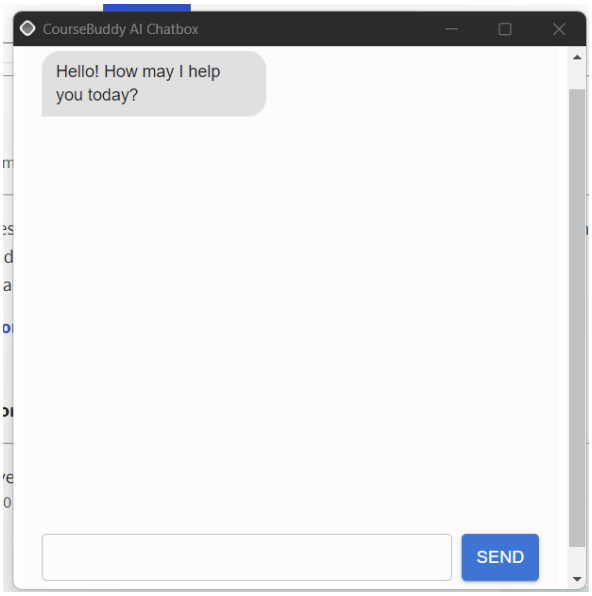

Figure 2A: Q&A Chatbox

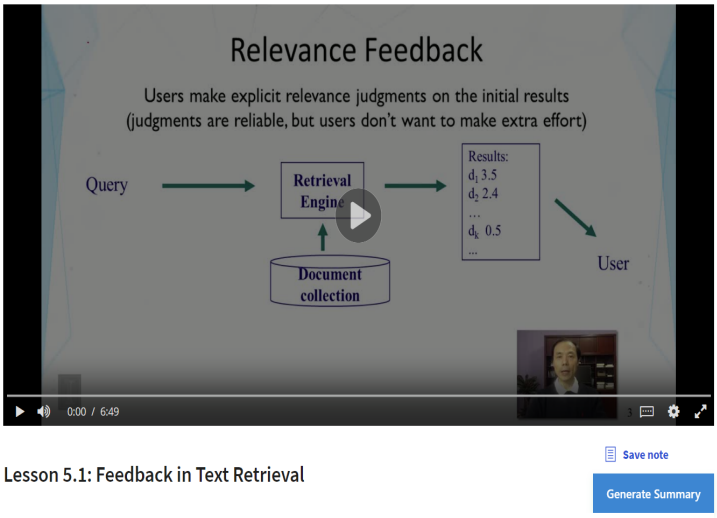

Figure 2B: Summarize Button

For the backend, we have developed a Python FastAPI application to provide a REST API for the frontend to communicate with. The implementation of all the core services have been completed including the File Upload Service, Summarization Service, Document Embedding Service, and RAG Chatbot Service.

The FIle Upload Service takes the uploaded lecture transcripts & slides (PDF) then persists them to a DigitalOcean Spaces bucket which is compatible with the AWS SDK (boto3). It also allows for the retrieval of the stored documents by the Summarization Service to generate bullet-point summaries of the lectures using the ChatGPT API. The summarization service uploads the summaries to the S3 bucket to avoid having to regenerate them again and also utilizes Redis caching to quickly check for recently completed summarizations.

Figure 3A & B: Summarization and FIle Upload Service

The Document Embedding Service then takes the uploaded documents and generates text embeddings using the HuggingFace (HF) BGE model which uses FlagEmbedding to map the lecture texts to dense-vectors (768 dimensions). This model has shown state of the art performance in the HF leaderboards for semantic search and is free for us to use (OpenAI's ada embedding model is not). After generating the vectors using this model, the embedding service stores them into a cloud hosted vector database which indexes them. This allows our RAG Service to quickly retrieve the relevant documents for a user's question based on the similarity measure (euclidean distance).

The RAG Service is used by our Q&A chatbot interface to provide the LLM 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. As the LLM takes some time to answer each question, we cache the responses to improve system performance so if the same question is asked again it takes less time to get a response. We also have investigated and implemented hybrid search for improved context retrieval by combining semantic search with BM25 search using the reciprocal rank fusion technique. However, we found that this approach is too complex for our use case and needs more fine-tuning with the parameters to perform well, but we have yet to complete the evaluation benchmarks required to do this.

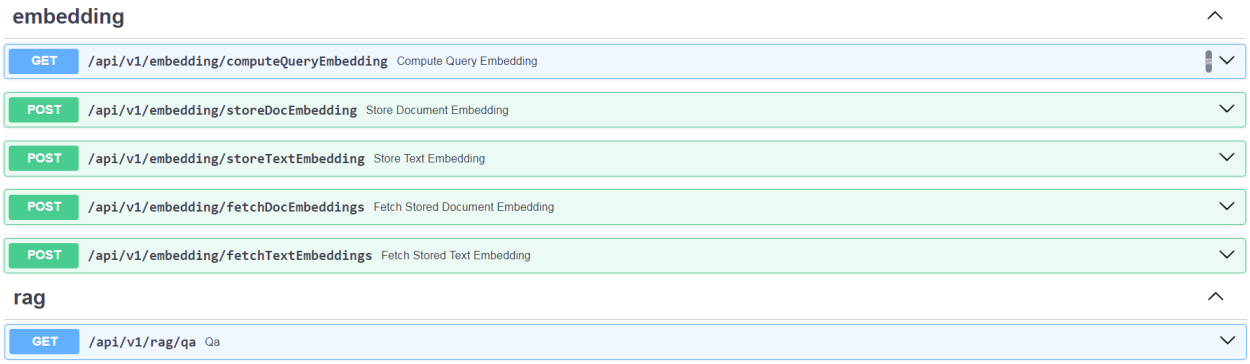

Figure 4A & B: Embedding and RAG Chat Service

For deployment of our backend REST Service, we already have most of the cloud infrastructure in place. This includes a serverless PostgreSQL database, the serverless vector database and a couple serverless Redis instances. Automated deployment to Google Cloud Run using Docker containers & Github Actions is still to be completed.

2. Which tasks are pending?

The main task currently being worked on is the frontend integration with the backend to link together and access the summarization, file upload and chatbot

services. We are also using this time to also conduct more thorough testing and fix any bugs that are identified during this process. Another task in progress is modifying the RAG Service to have the LLM provide citations to the sources of course content it used in generating the chatbot responses. However in some cases the LLM doesn't return the source, despite being prompted with the document IDs. We are doing prompt engineering and tweaking it to see if we can fix that inconsistent behavior but this is not guaranteed.

Our remaining tasks that are pending are mostly related to the final project submission. This includes writing documentation such as the user guide for installing and setting up the chrome extension. We also need to prepare a project presentation demo, but we first want to conduct final testing & performance optimization to ensure the useability and overall user experience is good. There might be some miscellaneous UI tweaks left to be done here and there, such as changing the UI style/appearance of the extension and adding some settings options to control certain features but that's all very minor.

3. **Are you facing any challenges?**

The main blocker we are facing is building an evaluation benchmark to evaluate the system's performance using a custom set of metrics. This requires first creating a dataset of user questions based on the documents that we have embedded in our database and then measuring how relevant the retrieved context is to the user question (retrieval precision) and whether the retrieved context is in the LLMs' answer (faithfulness / augmentation precision). To avoid manual effort we want to automate this by using the LLM to generate a dataset of questions and answers along with the relevant document used. While we have been able to manually do this once, we have encountered issues connecting to OpenAI when there is an outage and also quickly hit rate limits, thus preventing us from properly automating this. This issue has impacted the development of our other backend tasks also.

Originally, we planned to use the Azure OpenAI Service to avoid service outages and rate limits caused by public ChatGPT usage and had gotten free Azure credits to do so from the Github student developer pack. However it seems they are only granting access to enterprise users at this time as our application was denied. We have decided to stick to using OpenAI directly and paying for the usage, but still encounter such issues without a proper solution.

For the evaluation benchmark we were looking at implementing the metrics described in this paper titled "Automated Evaluation of Retrieval Augmented Generation," however the team member that was supposed to work on this suddenly decided to leave the team after not making much contribution or effort on the project. So

we likely will not be able to complete this task, given that the rest of the team is busy with the other tasks mentioned and we are running out of time.

For the frontend we have also faced different issues with the development of the chrome extension ranging from setting the correct permissions for our extension and general quirks with the Coursera website. One such example issue we ran into was integrating the chat box into the web page as the Coursera lecture video had a custom key listener for the “f” key that would override typing of “f” into the chat box and instead fullscreen the lecture video. Another challenge was learning the Plasmo framework which is the chrome extension framework we are using for development. Despite expecting it to make the development process easier, it has its own features that sometimes conflict with the default behavior of Chrome. The documentation is also not incredibly extensive, which makes problem solving such issues more time consuming.

For deployment of our backend service to Google Cloud Run, we are currently facing issues with the IAM permissions and getting it set up correctly to use the Github Workflow Actions pipeline. We may have to set up a new GCP project or change ownership of the GitHub repository to a different team member who has proper permissions.

#	Description	Staus	Estimated Effort (Hrs)	Time Spent (Hrs)	Comments
1	Define project scope and objectives.	COMPLETED	3	3	
2	Research and select appropriate development tools and frameworks.	COMPLETED	3	6	Frontend: React & Plasmo Backend: FastAPI & Langchain
3	Setting up the project infrastructure, including the development environment and project repository	COMPLETED	6	9	Using Github codespaces for development environment
4	Design the user interface of the Chrome extension, including file upload, summarization, and chatbot triggers.	COMPLETED	15	12	

5	Implement the file upload functionality to allow users to upload lecture transcripts and slides.	COMPLETED	15	12	
6	Implement text preprocessing to clean the uploaded text data (removing HTML tags, special characters, URLs).	COMPLETED	3	3	
7	Implement text summarization using LLM	COMPLETED	15	12	If there more time we want to add retry logic to handle OpenAI ratelimites
8	Implementing the text embedding to generate dense vectors from the uploaded documents	COMPLETED	5	5	Embedding service is ready, however for go-live we are not enabling embedding new uploaded documents.
9	Set up a vector database and implement indexing for documents	COMPLETED	3	3	Using Zillvus (Serverless Milvus)
10	Implement a Q&A chatbot interface using the Retrieval Augmentation Generation (RAG) technique.	COMPLETED	12	12	

11	Integrate a large language model (e.g. ChatGPT or Llama2) into the chatbot for answering questions.	COMPLETED	6	6	Originally we planned to use Azure OpenAI as we have free credit from the github developer pack. However it seems access is only being granted for enterprise users. We have decided to stick to using OpenAI directly.
12	Create a cache/database system to store responses for similar questions to improve system performance.	COMPLETED	3	3	We have implemented in-memory cache for the chatbot, however more effort than estimated is needed to add semantic caching
13	Investigate and implement hybrid search combining semantic similarity and BM25 search for improved context retrieval.	COMPLETED	6	8	This feature is implemented but will be turned off for go-live as it requires more effort for fine-tuning as we don't have an evaluation benchmark setup. Similarity search is performing well for our use case

14	Generate a dataset of questions and relevant context to evaluate the system's performance using metrics like MRR & NDCG.	BLOCKED	6	6	This step has been manually done but blocked due team member encountering issues productionizing
15	Develop a mechanism to provide citations to the sources of course content used in chatbot responses.	IN PROGRESS	3	3	This feature is still being tweaked. In some cases LLM doesn't return the sources, so citation is not possible. We are doing prompt engineering to see if we can fix that.
16	Conduct testing and debugging to ensure the system works as expected.	IN PROGRESS	6	6	Currently integration with backend and frontend is being worked on and bugs identified are being addressed
17	Document the project, including user guides and technical documentation.	NOT STARTED	3		
18	Prepare a project presentation demo.	NOT STARTED	3		
19	Conduct final testing, performance optimization, and quality assurance.	NOT STARTED	6		
20	Submit the project and prepare for presentation to the class.	NOT STARTED	3		
		Total (Hrs)	125	109	