LIGN 167 Final Project: Sample Practice Problem Generator

Nicholas Dai, Jason Kim, Christian Kumagai

1 Introduction

Students in class often take notes on key concepts so that they can review them before exams, but they often are not able to adequately create sample quiz questions to help them practice for the types of questions that will probably exist on exams. As it is often said that the best form of retention of knowledge is not merely just listening or note taking, but active application, students quizzing themselves is one of the best ways to prepare themselves for when they need to perform for exams.

This project allows students to use GPT-4o to create quiz questions. The project is a fairly basic UI combined with a fairly basic backend server that makes openai API calls in order to allow students to upload lecture files to generate quiz questions. Those quiz questions are generated with a fairly simple UI that allow students to actively test themselves to ensure they fully understand all the lecture details.

A flowchart is provided in Figure 1. This was borrowed from the project proposal.

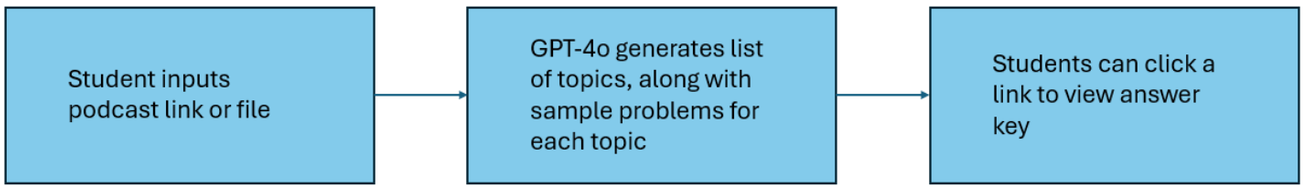

Figure 1: Flowchart for project

2 UI

The project was written in Javascript/Typescript, using the *npm* package manager.

This project was a single page web application, utilizing AWS's cloudscape as a means of design tools. The UI featured a fairly simple design. The static webpage when opened contains merely a button to upload a file, which can be either a .mp3 file or a .mp4 file, though .mp4 files do generally result in much longer upload times.

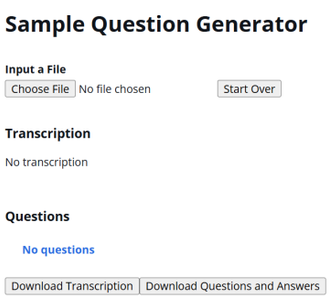

Figure 2: Opening Page

Upon upload, the frontend makes a call to the backend server, which will return both a transcript of the video along with the test questions generated from it. The user will be able to click on each test question to obtain the answers as well.

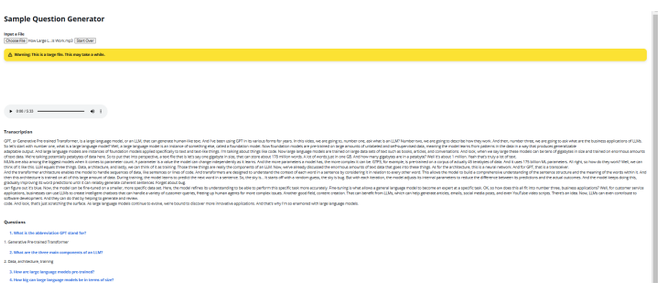

Figure 3: After Uploading Lecture

There also exists three other buttons. One is to download the transcript, one is to download the questions and answers, and one is to start over.

3 Backend Server

The backend of the server features a server written in Python. The only reason why Python was chosen was purely because it would be easier to write code using it.

The server accepts POST API calls from the frontend server, which sends an API request that contains the data from the lecture files. The server itself takes those requests and sends two calls to the OpenAI API. The first is to call OpenAI's

transcription, which transcribes the lectures into text, and the second is to call OpenAI's completion, which is essentially a standard chatbot that completes a request to write questions off of the lecture transcription text.

Upon receiving the transcription text, it is sent back to the frontend to be displayed.

A major difficulty of the backend server of this project was the fact that most lecture files were too big to be processed by OpenAI's API. In response, we used moviepy.editor and audioSegment from pydub to process files and to split them. This required downloading ffmpeg and configuring it to work with our server to actually allow files to be split into chunks for processing.

During the conversion process, all files are directly uploaded and downloaded from the local directory.

4 Future Development

Currently, the project itself still is not a standalone webapp. It utilizes merely localhost to function. However, in the future, this project would greatly benefit from the standard AWS four package architecture. The frontend would preferably have four packages: one for the UI, one for a CDK to host the site, and two Lambda packages where one could be used for external verification for using the webapp, and the other as internal verification for making API calls. The server itself could exist as a standalone package, although it would most likely benefit from being transferred to Javascript/Typescript to allow for proper CDK permission settings to allow it to run on the cloud.

With said type of architecture, all file conversions would likely be moved onto the cloud as well, as long term usage of this project (which currently uses local storage for file conversion) would simply result in storage being used up way too fast. AWS storage is quite expensive though, but it must be done to ensure scalability.

A major functionality improvement that could be made would be to allow for webscraping to obtain lecture files, so that lectures would not have to be uploaded through file upload but could be uploaded by submitting website links. However, after a few days of trying this out, we were not able to get it to work. Scraping Canvas for files is quite difficult, due to the extensive security the website has.

Additional improvements could include more abstract questions created from the website that involve actual problem solving. An example would be like generating a computational math problem from an abstract math concept.

Additionally, the UI could still be made to look a lot better. Our group made the mistake of not realizing we could use LLMs for code generation, so the majority of the code was hand written.

Regardless, there is much to improve on to improve the scalability of the program.

5 Conclusion

Overall, this project taught our group much about both LLMs, but also about software development and project management. We were able to work together to obtain a good working product.

A major point of education was learning how to use Python to set up a local server, and how to manipulate firewalls to allow for communication between multiple local servers at the same time. Building our own mini-API was quite fun.

In regards to LLMs, it took quite a while to figure out how to get the OpenAI API to actually generate text in the exact formatting we wanted it to, but it was quite rewarding once it was figured out.

All in all, this was a very educational and fun experience. Honestly one of the most fun projects I've done before!