AI Study Companion (Application) – Introduction to AI Final Project

Luke Evarretta

University of North Florida

evarretta.luke@gmail.com

Abstract

The motivation of this project is to create something that can be used by technophiles and overall users to augment learning by utilizing LLMs of the user’s choosing and an embedding model. In this project, I played around with numerous LLM and used the “all-MiniLM-L6-v2” sentence transformer to create a working web app that can potentially augment learning. This project resulted in the creation of a web app that has features such as explanation based on content, creation of flash cards based on content, and creation of quizzes with questions based on content. Depending on what LLM model is used, the response or the result of each feature can vary, with models of high parameters is preferred than thos with smaller parameters. Despite the capabilities of the produced web application, the real limitation of this project is that most end-users won’t have enough hardware resources, in terms of graphics, to enable the capabilities.

**Code** — https://github.com/MayDay5312002/AIBuddy---Intro-to-AI-Final-Proj

Motivation

The motivation of this project was to create something interesting, but impactful using some AI technology or technique. From brainstorming, I had numerous ideas on what I could have done for the final project, such as stock prediction, weather prediction, code analysis, and more, but I ultimately decided that I wanted to do something that is meaningful to me, and with a lot of college students. So, I decided that I wanted to create an AI study companion that would allow users to answer questions and get a response and also create automatic content such as flash cards and quizzes. Throughout my 4-year college experience, I have used numerous applications to allow me to succeed in my courses, such as Quizlet, Notion, etc., but nothing has had a major impact on my learning than ChatGPT. Just for disclaimer, I use ChatGPT for learning not to answer questions on a homework assignment, quizzes, or tests. I mainly use ChatGPT when I am in the process of learning some topic and I want to check if my understanding is in fact valid. Additionally, I use ChatGPT to avoid reading documentation of technologies. Sometimes it is a lot easier to enter a prompt in ChatGPT like “Is there a function in Python that would allow me to randomly shuffle the items of a list” than going to Google and searching for the same prompt. Although Google might also give you the right answer with a small amount searching, based on my experience with both, I am starting to realize that it is a lot easier to search something that is specific in ChatGPT than going to Google and searching it up.

When I decided to create this AI study companion, I wanted it to be somewhat different from ChatGPT. In terms of features, ChatGPT allows you to input a file and search the web to respond to a prompt regarding the content. You can also create an image with some diffusion model in ChatGPT. With all the features that ChatGPT has, there are few things that are missing in terms of what I think is needed for learning. When it comes to learning, I think having actual concise notes is a must have and having a feature that will quiz you on the content at hand is also something that should exist. So, I decided to have both of these features in my web application, along with the ability to explain things based on the content given by the user. I wanted this web application to be content driven, meaning you would have to input some file or YouTube video before entering a prompt.

Related Work

When researching for ways to develop this web application,

I came across an article from Medium, “Building an AI Study Buddy: A Practical Guide to Developing a Simple Learning Companion”, that does the exact same thing (Sanisetty 2025), except the technologies to develop the backend and frontend are different from mine. Also, the article is using LLMs through API, while I am using local LLM through Ollama, and I am also using formatted output to generate the flash cards and quiz questions, different from article where the author just parses the response without formatted output set up. In terms of features, like my web application, it is required to first input some valid file before entering a prompt and running the AI. After inputting such file, the users are able to summarize the extracted content, create multiple choice quizzes and flashcards based on the summaries.

A black background with white text

AI-generated content may be incorrect.There are also other applications that are developed for commercialized purposes like Mindgrasp AI, which is another AI study tool. With Mindgrasp AI, you can create summaries, quizzes, flashcards, and get answers to those documents you uploaded with questions in them. This is again commercialized, meaning you would have to pay on a monthly basis, unlike with mine and the article’s application, you just have to follow the steps to run it and have some study buddy that is free to use.

Methodologies/Algorithms/Approaches

Before starting this project, I had no clue what vector-based retrieval was and what they were used for, and with further research, I understood what their purpose is and what scenarios they can be used in. Despite the prowess of LLMs, they have their own limitations. For instance, when you give a really big input to an LLM, its performance to generate the appropriate response will most likely decline. This is due to the fact that LLMs usually just struggle with super long inputs. Technically, LLMs can handle big context inputs but they are less accurate, slower, and more random on those scenarios, especially without proper guidance with prompt engineering. This is where chunking and vector-based retrieval can be used to get better results with large inputs. In this project, I implemented chunking and vector-based retrieval to attack the issue of decline response with large inputs.

Here is the process of chunking and vector-based retrieval:

1. Chunk the document / split the document into smaller and more manageable parts.
2. Turn each chunk into a vector, that would represent the meaning of that chunk/piece. These vectors are we call “embeddings”. I am using the embedding model called “all-MiniLM-L6-v2”
3. Store these embeddings in a vector database (In the case of this project, we are using the vector database FAISS)
4. When the user enters a prompt, turn that prompt into a vector (embedding), find the top k chunks that are similar or related to that prompt, and feed the actual prompt and the chunks into the LLM

This process will lead to higher accuracy response because the LLM can now answer based on huge knowledge without getting lost. The benefits of this process are that it saves memory, makes the responses faster and also sharper, and allows LLMs to not lose focus on the important details of the prompt and the content.

A black background with white text

AI-generated content may be incorrect.After having the chunking and vector-based retrieval code, all we have to do now is prompt engineering and output formatting for flashcards and quizzes.

In terms of generating the flashcards, explanations, and questions of the quizzes, I had to ensure the prompts of each API endpoint for each feature are appropriate. For instance, the prompt for creating flash cards looks like this:

In this case, I had to try to guide the LLM into a response ideal for a specific prompt and content. In the screenshot above, notice how I gave necessary details. This is very important so that the LLM can generate the appropriate response with the necessary details. Without having concise but impactful prompts, the LLM could generate a response that is not accurate or that is not useful for the user. Furthermore, since we are using local LLMs, prompt engineering is even more important because as end-users we cannot use models of high parameters due to most of us having limited hardware resources to run models with high parameters. I decided to use local models to avoid having to pay for API access to certain models and to have full control over the data that is inputted into the LLM. Nowadays, data is easily commercialized to tech companies to better their products and services. For instance, you can use ChatGPT for free, but that could possibly mean that your input prompts could be used for training or other purposes. If you are heavily worried about security, it is better to run models locally, but this usually means you would have to buy the necessary hardware to enable LLMs to run smoothly and quickly. Moreover, I would also like to note that I have implemented thread process where the model can have some memory of previous conversations within that thread.

A screen shot of a computer program

AI-generated content may be incorrect.In terms of generating quizzes and flashcards, I created models to describe the output for these two endpoints. A model would look like this:

In the screenshot above, we have to two models’ “FlashCard” and “FlashCards”. “FlashCards” would have a list of “FlashCard”. I decided to set it up this way so that the LLM can generate numerous flash cards, depending on the input number of flashcards, with one run of the LLM. After having these models’ classes created, I then passed it to the chat option called format.

This is where the LLM decides what content is inserted for each attribute of the model. It is important to note that having appropriate attribute names and model names is also important so that the model can decide what to put in what model and what content to put in each attribute. In addition to these flash card models, I also have two models for quizzes with the same idea. I have “QuizCards” and “QuizCard”, where “QuizCards” would have a list of “QuizCard”. Furthermore, the output of a run with a model would be a formatted output that follows the attributes of a model, and it would like in JSON format, which is ideal for responding to client request and creation of flashcards and quizzes.

Experiment and Results

During testing of my AI study companion, I utilized numerous models, ranging from 1B to 8B parameter models. I have come to a conclusion that models with higher parameters usually have more impactful responses. For instance, when I was using the model “phi4-mini:latest” which has about 3B parameters would sometimes respond with content unrelated to the prompt, but with “Llama3.1:8B” the output is a lot more accurate and more impactful and will more likely follow the prompt. I recommend utilizing models with higher parameters but if your system cannot handle the load then choose the appropriate models best for your system.

Moreover, I also tried beforehand not utilizing vector-based retrieval and simply giving the whole content, and I observed that the responses are more likely to lose track of the given content and prompt. I also observed the generation of response becomes slower. I hypothesize this is due to the large data that was inputted. This is where I realize the significance of vector-based retrieval or embedding retrieval. A lot of the time the LLM does not need the whole document to answer a prompt, it just needs certain parts or chunks to generate a response.

Conclusion

Throughout the development of my AI study companion, I truly enjoyed the process, and I am glad that I went through it because I learned techniques, such as chunking and vector-based retrieval and output formatting. Although I would not consider this web application finished, even though it is a working and capable web application, I would recommend people to utilize it or at least learn the process within. Vector-based retrieval is truly an enhancement compared to just simply inputting the whole content. Not only does it lead to quicker response, but it also leads to a more accurate response.

References

Sanisetty, H. 2025. Building an AI Studdy Buddy: A Practical Guide to Developing a simple learning companion. *MEDIUM*. https://pub.towardsai.net/building-an-ai-study-buddy-a-practical-guide-to-developing-a-simple-learning-companion-550daaa58640

LangChain. 2025. LangChain Python API Reference. https://python.langchain.com/api\_reference/. Accessed: 2025-

04-26.

Mindgrasp AI. 2025. https://www.mindgrasp.ai/. Accessed: 2025-

04-27.