Collin Jung
CS 224G

Final Reflection

Coming into this project, I had very little background in finance and did not exactly know what I was getting into. For the first few weeks, I spent some time looking into what issues we were trying to solve and what exactly was needed from a software perspective for our audience. DiligenceDynamics was our solution to solving inefficiencies in a finance setting in which time is currently wasted with parsing documents. Specifically, we found that a lot of the time, analysts are flipping through documents and tables to extract data related to earnings and quarterly results. In order to combat this problem, we wanted to be able to store documents and pull based on user queries to make this process more efficient and automated.

A takeaway that I had through pursuing this project was the power behind data embedding, especially with multimodal embeddings. One of the early issues we had was that tables and images were not easily read, or sometimes not even registered at all, by a standalone language model. Since the data we were working with was money, we could not afford to lose accuracy when it came to tables of data, a common data structure for finance documents. Thus, we decided to incorporate a multimodal embedding system in which all data formats were converted into vector embeddings based on their content. This allowed us to be able to pull documents based on their content without running into our initial issues.

Another thing I learned was how to incorporate RAG models with storage systems. In the early stages of our project, we stored documents for our RAG model locally. However, this quickly became an issue because it was unscalable for larger scale projects and lots of the data we needed later on could not be stored easily. To solve this issue we switched to Firebase, which allowed us more flexibility.

When working on the front-end, I ran into some other challenges related to data visualization and API calls. I designed the early prototypes for our company search page, which pulls finance data from an API and shows stock prices and recent news. Once I set up the API for the finance data, I realized that I would have to find a way to graph the data like it looks on existing stock information websites. For this, I was able to look into native graphing libraries in Javascript to show the change in stock prices and also implement functionality for changing dates manually. For the News API, I pulled articles based on the company name and ticker and parsed through the top 5 most relevant articles.

Finally, for our chatbot, I was able to come to some takeaways related to prompt engineering and guard rails. For our investment report copilot, I needed a way for the chatbot to refer back to existing documents in the backend while also responding in a way that was geared towards generating reports. Thus, we decided to use the same model for both the company search chatbot and the investment report copilot. Using the same model and switching the prompt based on where we were calling the model meant that we could use the same RAG functionality, just in a different way based on the user's needs. Of course, this had its limitations in that we had less flexibility and the model could not pull enough data to formulate its answer. To combat this, we raised the maximum number of documents retrieved for the investment report copilot to allow for more detailed responses.

DiligenceDynamics allowed me to explore RAG models in more detail while also being good practice for implementing full stack web apps. I think that working in this startup-esque environment has been a very insightful experience and will definitely use the concepts learned throughout this quarter in my future projects.