Winston Shum
CS224G

Individual Insights Report

In building the DiligenceDynamics web-app, there were many insights gained from building the underlying RAG model for our question-answering chatbot and investment report copilot.

First, one key takeaway from working on the RAG model and making it more efficient in retrieving from a large corpus of documents is to first break the document down into smaller chunks of texts/tables/images, and then generate summaries on these smaller chunks. Importantly, compared to embedding an entire PDF document or raw text (some of which were over 100 pages long in our case), only creating embeddings of the summaries of these smaller chunks allowed greater retriever accuracy and quicker text generation in down-stream tasks, allowing our Q/A to output an answer to user queries in a timely manner. Further, another important part of our RAG model was the contextualization of queries. To allow the user to have an intuitive chat experience with our chatbot, we wanted to ensure that users don't always have to provide a specific question/query to the chatbot, and instead, the chatbot should be able to infer context from the chat history. The naive solution here would be to simply pass the chat history into the prompt for the LLM. However, this ended up drastically increasing the prompt length and made the chatbot focus on the chat history instead of the context documents retrieved by the retriever. To fix this problem, we came up with an intermediary step: to have an LLM reformulate the query based on the chat history first. While this means that we have two separate calls to the LLM, the resulting response from the LLM was substantially more accurate and improved the conversation flow. Finally, one last challenge that we faced was that we wanted to use the same RAG model for both the chatbot and the investment report writer feature. While changing the prompt and providing an example investment report helped the LLM understand how to write a report, the generated report was often too short and did not provide enough detail. The key insight here was that the retriever did not retrieve enough context. Specifically, as a chatbot, the RAG model would retrieve the top 4 documents. However, for the investment report copilot, given that we wanted more exhaustive explanations and descriptions, we ended up modifying the retriever to use the top 8 documents, which improved the level of detail in the generated investment reports while maintaining accuracy.

Besides the RAG model, one of the key challenges that we faced was how to set up the backend databases that store the actual PDF documents along with the vector embedding database. While we originally started with a local PDF database and Chroma for our vector embeddings, we quickly realized that storing them locally and on Chroma was not a scalable solution, causing us to switch to Firebase and Pinecone. However, even after we switched platforms, we realized that we had to store a lot more metadata than we thought and had to keep resetting our database to include metadata such as user IDs, time of upload, namespaces, etc. Again, the insight we gained from this is that we should have thought about how users would interact with our applications before setting up the backend databases. For example, given the amount of proprietary data that exists in the finance industry, it would be important to separate the embeddings and documents based on individual user IDs. Further, given that users typically would want to research a specific company or industry, we should have separated embeddings and documents by company name as well. This costs our team a lot of time just resetting and re-uploading documents to our databases for each small change in metadata.

Overall, building DiligenceDynamics was a great learning experience for me, both from a technical and a business standpoint. It helped me gain both front-end and back-end coding experience while also realizing how our code should ultimately be tailored to user needs.