First Task: Document-Based Question Answering

Evan Hutchins

Introduction: One of the main challenges faced with building a document-based question answering system is effectively processing long sequences of text. LLMs are a natural solution for this knowledge-intensive work because of how these models excel at dealing with natural language. However, these models suffer from hallucinations, especially when working with longer-form content. After surveying several papers, I found some common design patterns to address this issue, and successfully built a prototype to showcase these findings.

Literature: The primary technique to solve these problems is through Retrieval-Augmented Generation (RAG), which has quickly become a popular technique in document-based question answering (Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020)). RAG combines a pre-trained generator like an LLM with a pre-trained retriever that fetches relevant documents so that the model can respond based on both a query and the retrieved context, which is very useful when processing longer documents. Instead of processing an entire document to answer a question, our model can instead retrieve the necessary information and then build a response based on the given context.

For example, PaperQA proposes a RAG agent for answering questions over scientific literature (Lála, J., O'Donoghue, O., Shtedritski, A., Cox, S., Rodriques, S. G., & White, A. D. (2023)). PaperQA utilizes multiple features like RAG models and three LLM agents. First, in the search phase, the system searches for papers relevant to the query, and then chunks and embeds these papers. Next, in the gather evidence phase, the system embeds the query into a vector, and ranks the top document chunks using MMR vector retrieval for more diverse context. Then, it creates a scored summary of each chunk using an agent LLM. Finally, in the answer question phase, the system generates an answer using the best summaries as context. If the answer is sufficient, the agent LLM answers the question, otherwise it repeats the process to find more evidence.

After experimenting with a dataset and comparing PaperQA to other popular models as well as human performance, PaperQA answered questions about scientific literature at a similar accuracy as humans, without any hallucinations. In their latest paper, the newest model PaperQA2 has exceeded human performance and can even identify contradictions within scientific literature (Skarlinski, M. D., Cox, S., Laurent, J. M., Braza, J. D., Hinks, M., Hammerling, M. J., ... & White, A. D. (2024)).

Implementation: I based my model on the workflow outlined by PaperQA; however, I simplified the model because PaperQA was designed to combine information from many papers, while I intended to create a model from a single paper. Also, I decided against using multiple agents for this prototype to keep the model simple and just build a simple RAG agent using a single LLM. I built my prototype in Google Colab; you can view my code here.

The first step of my prototype was to embed a pdf document in a vector database for an LLM to later retrieve chunks for context in its response. Using PyPDFLoader from LangChain, I loaded a pdf document and extracted all of the text. Next, I split the document into chunks using LangChain's RecursiveCharacterTextSplitter with a chunk size of 1500 tokens and chunk overlap of 150 tokens. Finally, I embedded the chunks in a Chroma vector database using the *all-MiniLM-L6-v2* embedding model from Hugging Face.

The next step was to retrieve chunks from the vector database to use as context for the LLMs final answer. I created an answer function that takes a question, a vector database, and an LLM as parameters, and outputs an answer to the question in natural language using LangChain's RetrievalQA chain. For this, I created a retrieval method using MMR vector retrieval as done in the PaperQA model for more diverse retrieval of relevant chunks. It retrieves 10 chunks, and then returns the top 5 diverse chunks as context. I also rewrote a prompt used by the answer LLM in PaperQA's model for this prototype. I tested my prototype with two LLMs, one run locally on Google Colab, and one through Google's Gemini API. On Colab, I used Hugging Face's *flan-t5-large* running on Colab's T4 GPU. From the Gemini API, I used *gemini-2.0-flash*.

Limitations: One of the main limitations that I ran into when building my prototype were resource limitations. The free version of Colab could only handle very lightweight models, so I had to use a small model that could only use a very small number of tokens. As a result, the final output often repeated sentences or generally seemed confused by the question. To avoid these resource limitations, I decided to try out Google's Gemini API and was amazed by how accurate the results were. Gemini 2.0 Flash allows a set number of 1,000,000 tokens per minute and 1,500 requests per day for free, but this was more than enough for me to test my prototype.

I tested the prototype on the PaperQA pdf document by embedding the pdf and then asking the model questions about it. This prototype was very strong at answering questions about different sections of the paper. I started by asking for very broad summaries, and then asked follow up questions about specific sections, and the prototype was able to correctly pull specific information from the paper. The model was

also able to correctly answer questions about a table in the paper, answering questions about how the PaperQA performed compared to different LLMs. The model also successfully pulled relevant citations within the paper when asked about similar papers about RAG agents.

One type of question that it could not correctly answer was basic trivia like the full name of the paper and the authors of the paper, it just returned that it was unable to answer the question. Another limitation is that this model cannot hold previous conversations in memory. This is mostly a limitation of the RetrievalQA library, which does not automatically hold chat history as context. While this is not explicitly necessary for a model like this, it would definitely improve the user experience. Along a similar line of thinking, the model was unable to reason or make any assumptions based on the findings of the paper other than inferences made within the paper.

Overall, I think that this prototype works well enough for production use. It is able to accurately answer questions about documents, and could be built into a production system. Most improvements would be to add new features to improve the user experience.

Improvements:

Most of the improvements to address the limitations will add new features, but there are some important parts to address for the base model. To ensure that the model can answer basic questions like paper title, authors, and publish date, I could extract this information separately and add this to the context that is given to the final answer LLM. To build conversation memory into this model, I can use the LangChain ConversationalRetrievalChain library to build on top of the RetrievalQA chain. This would allow users to ask more specific follow up questions about specifics of a paper.

One improvement that I also want to try to make is to enable the model to reason about the findings of the paper and allow for more discussion between the model and researchers to allow researchers to further develop new ideas based on the current paper. This is a much more ambitious idea, but I think one strong start would be to fully implement the PaperQA model, including multiple agent LLMs and a search feature to query from other similar papers. I also want to look into other LLM models to see if there are other models more suited for reasoning that could be included as a separate agent within the system. Overall, I think that this work has lots of applications, not just in research but also for understanding large documentation or even source code with some modifications.