A State-of-the-Art Retrieval-Augmented Generation Framework for Accurate Document-Centric Question Answering

Vedik Upadhyay

Introduction

Document Question Answering (DocQA) enables Large Language Models (LLMs) to answer user queries using structured or unstructured knowledge sources. [6] While LLMs excel at language tasks, they suffer from hallucinations and outdated knowledge banks, especially in domains where data changes real-time or special cases like handwritten documents [3, 4, 5]. Retrieval-Augmented Generation (RAG) addresses these limitations by integrating retrieval mechanisms to assist LLMs by providing them relevant context. This paper combines state-of-the-art DocQA frameworks and retrieval methods to create a RAG pipeline for accurate document question answering, mainly for text-based documents such as pdfs.

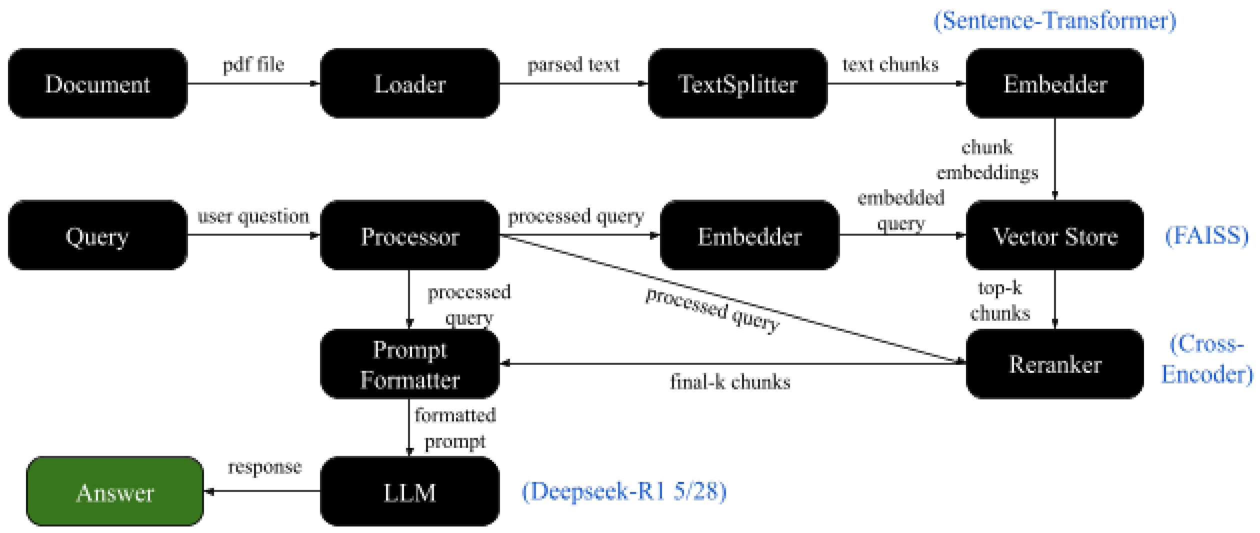
Overview of Related Works

The core RAG framework [1] which this model is based on combines a retriever (such as BM25) and a generator (such as T5). Given a query q, the retriever fetches relevant documents or passages x from a knowledge base, and the generator produces an answer a conditioned on q and x. Traditional vector retrieval, which is typically dense in nature (meaning that vector embeddings have a low number of dimensions, and each value is meaningful) struggles with sparse documents such as short tweets, messages, or texts. To address this challenge, dual-path retrieval [3] combines semantic retrieval (like M3E, dense), lexical retrieval (like BM25, sparse), and reranking modules like BGE to boost precision and accuracy. Retrieval was also specialized when performing QA on handwritten documents [5], by combining the individual strengths of the Term Frequency–Inverse Document Frequency (TF-IDF) algorithm with sentence transformers for better retrieval accuracy.

Many different papers on DocQA propose RAG-based frameworks to achieve enhanced performance in many different situations. Researchers used a retriever-generator framework with open-source LLMs to effectively train their dense retrievers while addressing privacy concerns raised by cloud-based LLMs. Tuning and using the generator allowed them to achieve better overall results compared to plain dense retrieval or plain RAG [4]. They also used Chain-of-Thought (CoT) Fine-Tuning with the LLM, instructing it to summarize documents, consider relevance between query and context, and only generate answers from provided documents, which reduced hallucination in results. UniGen [2] proposes a framework that utilizes LLMs in both the retrieval and question answering stage of their pipeline. Using an LLM to expand queries (Q-Connectors) and summarize relevant documents (D-Connectors), the model outperformed baselines and bridged semantic gaps via its connectors. The model also used iterative enhancement, which uses previous answers to refine subsequent inputs for a more accurate final answer. This technique was possible because of the generative-nature of their framework.

Method

The method that I decided to pursue and deemed as best is very similar to the one proposed in “Enhancing Large Model Document Question Answering through Retrieval Augmentation” [3]. The workflow I implemented is as follows, and you’ll find it’s almost identical to the one outlined in the paper:



First, the document, which is normally a pdf, gets parsed, normalized, stripped of any white text/special characters, and stored into the model. The parsed text then gets split into chunks by a text splitter, and they are then embedded into matrices using a Sentence Transformer. The paper uses M3E as an embedder as it is a common practice, but for my use case (no need for multilingual support), I used a sentence-transformer for speed and size. The embeddings are then added to the vector store, which in this case is Facebook AI Similarity Search (FAISS). Meanwhile, the user’s query gets processed by a processor to get rid of whitespace and fix spelling mistakes, and then FAISS uses its index and the embedded query to perform an Exact Nearest Neighbor search using inner product, retrieving the top-k relevant chunks. The top-k chunks are then reranked by a cross-encoder based on their relevance to the query to produce the final-k chunks. The reranker from the paper utilizes both semantic and lexical match, while I use a cross-encoder. Cross-encoders are optimized for semantic matching, capturing meaning at a deeper level by jointly encoding query and context pairs. Although cross-encoders have implicit capabilities to match the same exact tokens/words, they don’t explicitly use term-weighing algorithms like TF-IDF, and instead score the semantic relevance of pairs from context picked up during their training. Lastly, to generate an answer from the LLM, the query, final-k chunks, and a system prompt with clear CoT instructions are formatted into a prompt and sent in an API request to DeepSeek-R1. I chose to use R1 because of its reasoning capabilities, and on-ppar performance with GPT while being very cost-efficient (openrouter provides 50 free API requests per day).

I determined that this solution is the best solution because it is practical yet extremely efficient. Libraries like FAISS and LangChain allow for seamless integration of similarity search, vector store, and text splitter, yet are highly effective in their respective tasks, allowing me to better reproduce state of the art (SOTA). Similarly, embedding and reranking can be achieved using free-to-use, lightweight models like cross-encoders and sentence-transformers without sacrificing too much accuracy. Overall, this model was made using only Google Colab's computing power and cost $0 to make (including API requests), making it highly cost effective while staying very accurate. For practical use, that makes it deployable and friendly to the everyday consumer, and for those reasons I believe this model is very useful and I would use something like this as a product. Additionally, its models are replaceable (e.g. replace DeepSeek with GPT4.5) so the solution can be scaled for enterprises or large-scale use.

Performance

In terms of performance, I believe that this model performs extremely well to the point where I would even use it for my DocQA needs. Using "train-v2.0.json" from the Stanford Question Answering Dataset (SQuAD), I extracted 37 random questions (not hand-picked, randomly selected by program) and ran my model on all 37 of them. Out of 37, 35 were answered correctly, achieving a 94.6% accuracy in that domain of question answering. The 2 incorrect answers were cases of hallucination, where the model gave answers to unanswerable questions, meaning the model had a 100% accuracy rate for answerable questions. Although only a small subset of the dataset was used (due to cost limitations and overwhelming size of dataset), I still think it was a very impressive result, considering how fast the model read the context for and answered 37 questions.

Caveats and Potential Limitations

As I was building the model, I also became aware of certain limitations:

1. Increasingly complex queries - as queries become more complex, it becomes harder to retrieve all relevant chunks and also process the query properly. Compound questions or questions that require deep understanding + synthesis of different parts of the text may be harder for the model to answer effectively when compared to a human. A potential solution to this could be query expansion for better context + retrieval, similar to what was detailed in the UniGen framework [2].
2. Hallucination & Chat History - my model provided answers to unanswerable questions, however I noticed that part of the reason it did that was because of words like "it" and "that," which referred to previous questions in the question set. When the model doesn't know what those pronouns refer to, it can wrongly assume and hallucinate answers. A potential solution to this could be including chat history into the formatted prompt, however this would only work as long as the context doesn't exceed the model's limit. DeepSeek-R1 can take up to 164,000 tokens.

References

1. Lewis, P. et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. NeurIPS. https://arxiv.org/pdf/2005.11401
2. Li, X. et al. (2024). UniGen: A Unified Generative Framework for Retrieval and Question Answering. AAAI. https://arxiv.org/pdf/2312.11036
3. Zeng, J. et al. (2024). Enhancing Large Model Document Question Answering through Retrieval Augmentation. IEEE AIPS. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10710053
4. Jiang, F. et al. (2024). Enhancing Question Answering for Enterprise Knowledge Bases using LLMs. ACM SIGIR. https://arxiv.org/pdf/2404.08695
5. Pal, A. et al. (2025). Advancing QA on Handwritten Documents: A SOTA Recognition-Based Model. [Preprint]. https://arxiv.org/pdf/2406.17437
6. Caballero, Michael/ (2021). A Brief Survey of Question Answering Systems. IJAIA. https://aircconline.com/ijaia/V12N5/12521ijaia01.pdf