Indian Institute of Technology (Indian School of Mines), Dhanbad

Enhancing Retrieval and Privacy of RAG-based Question Answering Models on Documents

Riya Kumari Jha, Ayush Kumar, and Kumari Anjali

November 7, 2024

Contents

1 Problem Statement

The use of Large Language Models (LLMs) is increasing rapidly, as they provide advanced capabilities that help developers and businesses to create solutions for complex, real-world tasks. However, commercial LLMs present key limitations related to privacy and cost-efficiency. Models like GPT-3 or GPT-4 are static in knowledge. They are not continuously trained and are capped by their knowledge cutoff (e.g., GPT-3's data goes up to 2021). This poses challenges for applications requiring up-to-date or context-specific answers, such as personal finance advisories, medical guidance or proprietary technical details.

Existing models also suffer from hallucination (generating confident but incorrect answers) and struggle to deliver contextually specific, reliable responses. This makes them unsuitable for precise, domain-specific queries, like recommending specific medications or interpreting proprietary financial data. Moreover, using cloud-based LLMs introduces privacy risks, as sensitive data, such as medical information, might leave secure, controlled environments.

To address these challenges, Retrieval-Augmented Generation (RAG) models have gained attention. RAG combines a retieval mechanism with a generative model, allowing the LLM to leverage external, up-to-date data sources, which mitigates knowledge cutoff issues and reduces hallucinations. However, current RAG implementations are often cloud-based, adding to the privacy risk and increasing costs due to ongoing cloud computations and data retrieval fees.

1.1 Limitations of Current RAG Pipelines

In summary, the current RAG pipelines suffer from the following limitations:

- Domain-Specific Accuracy: LLMs struggle to answer domain specific based questions accurately, often generating overly broad or incorrect responses.
- Privacy Risks: Sending sensitive data to cloud-hosted LLMs exposes it to potential breaches.
- High Operational Costs: Cloud-hosted RAG models incur continuous data retrieval and compute costs. Processing RAG locally can substantially reduce these expenses.

In light of the above limitations, this project work aims to enhance the retrieval accuracy and privacy of RAG-based Question Answering (QA) models deployed on local infrastructure. As the requirement of these enhancements is very crucial in industries like healthcare, we currently intend to use this enhanced RAG-based QA model on a single human-nutrition document which could later be extended to multi documents in different industries too (like financial industries).

1.2 Project Objectives

The objectives of this work aims to achieve the following:

- Domain-Specific Accuracy: Refine retrieval and generation to yeild specific, accurate answers tailored to human nutrition domain-specific queries, reducing hallucinations.
- Enhance Privacy: Implement a local RAG pipeline on a GPU to keep sensitive data secure within the user's device, avoiding potential privacy risks associated with cloud-based models.
- Cost Efficiency: Leverage local processing to minimize recurring cloud-based costs, making RAG models more sustainable for organizations.

Hence, this project work is divided in 2 parts. The first half of the project focuses on the enhancement of retrieval accuracy and the other half focuses to enhance the privacy of existing RAG pipeline.

2 Workflow: Current RAG Pipelines

Most RAG pipelines first break the collection of documents into uniform chunks without taking into consideration of the documents' structure or content. Each chunk is made up of same number of tokens or words. These chunks may or may not have overlaps in the context, which means the last line of the previous chunk is present in the first line of the next chunk. The retriever then returns the top k most similar chunks, and each chunk will be of approximately the equal size. The drawback of this approach is that it does not take into consideration of the nature or context of the document which may lead to information loss in certain situations. In these scenarios, a traditional RAG pipeline would not provide the most ideal chunk.

Even the cosine similarity which makes the core part of RAG suffers from challenges in retrieving relevant information. This is because the most similar chunks may not be the most relevant chunks for certain use cases.

When dealing with single documents, basic chunking techniques (with fixed chunk size) or similarity searches might work well but with the increasing number and complexity of documents, these basic techniques might perform poorly.

Standard retrieval methods also fall short in complex question answering. These algorithms primarily rely on direct similarity matching, missing the logical reasoning steps which could be involved with it.

3 Workflow: Enhanced RAG Pipeline

The proposed workflow for this project can be explained with the help of the below diagram.

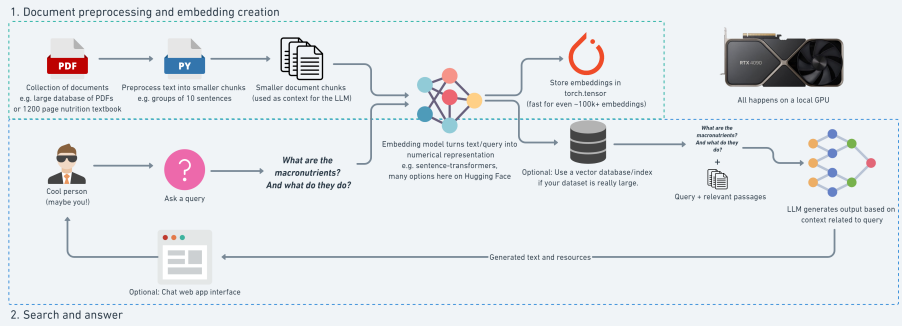

Figure 1: Workflow Flowchart

The retrieval process involved in our enhanced RAG-pipeline remains mostly the same but we will need to work on to improve the accuracy of the retrieved results. We will explore a variety of strategies to enhance the retrieval accuracy such as chunking techniques, query expansion using HyDE (Hypothetical Document Embeddings), re-ranking algorithms and fine-tuning of embedding algorithms. This constitutes the first part of the project.

The other part of the project would focus to improve the privacy of the existing RAG pipeline. Once, we get our retrieval pipeline read, we will start with the generation part. We will be using a Large Language Model (LLM) to generate an output for a given input based on the context of relevant information to the query. This input to the LLM is referred to as a prompt. An important part here will be to choose a suitable LLM keeping in mind that LLMs with higher number of parameters perform better but also a larger size model may or may not be able to run on our available hardware (GPU). Once this is decided, we can generate text with out local LLM model by passing a tokenized input. With this we will be able to generate domain-specific output from

an LLM running locally. Hence, we will be able to cover the R (Retrieval) and G (Generation) of RAG.

To cover the last step of Augmentation (A), we will put together our search for relevant chunks and a prompt that we will pass to our LLM. This is popularly known as prompt engineering. This completes our RAG pipeline Retrieved, Augmented and Generated and all this will be on our local GPU.

4 Implementation: Retrieval Accuracy Enhancement

Considering the limitations of the traditional RAG pipelines, we clearly see a need to improve the retrieval accuracy of these existing pipelines. To do so, we have a few strategies that aim to optimize RAG performance for our use case of answering questions and chatting with human nutrition document.

4.1 Improvised Chunking Technique

This method is a crucial step in retrieval process. This is because having too much irrelevant data creates noise and may affect the retrieval output. Most RAG pipelines, choose an appropriate chunk size in this process but this may lead to context-loss as too much or too little information in the chunks may affect the output accuracy for a query. We therefore, need to take into consideration these 3 points to improve retrieval accuracy through chunking:

- nature (e.d., short or lengthy) of the documents in our knowledge base
- capacity of our embedding model (e.g., token limits)
- a potential range of the length of the queries the model expects

We performed a rough exploratory data analysis to get an idea on the size of the texts (e.g. character counts, word counts, etc) we are working with. The different size of texts will be a good indicator into how we should split out texts. From different embedding models, we chose 'all-mpnet-base-v2'. This embedding model has a token limit of 385 tokens. This means that the model has been trained to ingest and turn into embedding texts with 384 tokens (1 token = 4 characters = 0.74 words). Texts over 384 tokens which are encoded by this model will be automatically reduced to 384 tokens.

The results of this analysis on our data is displayed in the table below.

	page_number	page_char_count	page_word_count	page_sentence_count_raw	page_token_count
count	1208.00	1208.00	1208.00	1208.00	1208.00
mean	562.50	1148.00	199.50	10.52	287.00
std	348.86	560.38	95.83	6.55	140.10
min	-41.00	0.00	1.00	1.00	0.00
25%	260.75	762.00	134.00	5.00	190.50
50%	562.50	1231.50	216.00	10.00	307.88
75%	864.25	1603.50	272.00	15.00	400.88
max	1166.00	2308.00	430.00	39.00	577.00

Figure 2: Exploratory data analysis

From the above table, the average token count per page is 287. This means, for our use case, we can embed an average whole page with the 'all-mpnet-base-v2' model with an input capacity of 384.

We have done a further pre-processing of our data before embedding it. We used a simple approach of breaking the text into chunks of sentences for our use case. This would make handling larger pages of text easier and we can get specific group of sentences used within our RAG pipeline. A page of the document is chunked into groups of n number of sentences. With several hit and trials, a chunk size of 10 best suited our data.

There were few options to split text into sentences, namely:

- split into sentences with simple rules (e.g. split on ".")
- split into sentences with a natural language processing (NLP) library such as spaCy

We preferred spaCy to break our text into sentences since it is likely a bit more robust than just splitting on full-stops. However, from the stats, it is observed that the page-sentence-count from splitting on full-stops is quite close to that from spaCy.

	page_number	page_char_count	page_word_count	page_sentence_count_raw	page_token_count	page_sentence_count_s
count	1208.00	1208.00	1208.00	1208.00	1208.00	12(
mean	562.50	1148.00	199.50	10.52	287.00	
std	348.86	560.38	95.83	6.55	140.10	
min	-41.00	0.00	1.00	1.00	0.00	
25%	260.75	762.00	134.00	5.00	190.50	
50%	562.50	1231.50	216.00	10.00	307.88	
75%	864.25	1603.50	272.00	15.00	400.88	
max	1166.00	2308.00	430.00	39.00	577.00	

Figure 3: Page Sentence Count comparision between raw splitting and splitting using spaCy

It is also observed from the table that we have an average of 287 tokens per page. On average, each of the page in our document has 10 sentences. Hence, our groups of 10 sentences will also be 287 tokens long. This gives sufficient room for the text to be embedded by our 'all-mpnet-base-v2' model (with 384 token limit).

The chosen values of the above listed parameters for our use case of human nutrition document is shown in the table below.

parameter	**value**
nature of documents	short
chunk size	10
embedding model	all-mpnet-base-v2
embedding model token limit	384
approx. length of queries	10-15 tokens per query

Table 1: Parameters considered for chunking

4.2 Query Expansion using HyDE

Query Expansion is a method to alter the query given by the user which is then fed to the RAG pipeline to find relevant chunks. We make use of Hypothetical Document Embeddings (HyDE) which uses an LLM to generate more similar queries using synonyms of the words in the original user provided query. It then performs a similarity search with both the hypothetical queries that it generated and the original user provided query. Although, it is a good method but this is highly use case dependent. Hence, for our use case, we found that the accuracy scores of the chunks received decreased significantly using this method. This can be because in the process to generate

hypothetical similar queries by making use of the synonyms of the words from the original query, the actual context of the query was lost.

4.3 Re-ranking Algorithm

Traditional RAG pipelines provide the flexibility to the user to choose the number of chunks to include as context for a query, typically selecting the top 1-2 chunks based on similarity measures like cosine similarity. However, these similarity-based algorithms return the most similar chunks and not the most relevant ones.

To improve relevance, re-ranking algorithms prioritize meaningful context over just similarity. Cohere's re-ranking algorithm is one approach that enhances relevance by leveraging machine learning and NLP techniques beyond basic similarity searches.

However, we were not able to get satisfying results on using Cohere's re-ranking algorithm.

4.4 Fine Tuning Embedding Model

Embedding algorithms convert the texts into useful numerical representations. Embeddings are learned representations. This means rather than directly mapping words/tokens/characters to numbers directly (e.g., "a": 0, "b": 1, "c": 3, ...), the numerical representation of tokens is learned by going through large corpuses of text and figuring out how different token relate to each other. Ideally, embeddings of text will mean that similar meaning texts have similar numerical representation. We can fine-tune embedding algorithms based on human nutrition domain-specific knowledge to enhance retrieval in this domain. Embeddings adapt themselves when words have slightly different meanings based on the context.

In this project, we have fine tuned our embedding model all-mpnet-base-v2 to improve the retrieval accuracy. We prepared a dataset of 100 "Questions", "Answers" and "Relevance Score" and used the sentence transformer library to fine-tune our embedding model. This library allows us to train embedding models with minimal effort and provides different training methods for this purpose. With this method, we were able to see a slight improvement in the accuracy scores of the relevant chunks.

5 Evaluation

This section discusses the evaluation of the retrieval outputs using all the above techniques. This is done by taking into consideration 2 points:

- Retrieval Quality: how effectively the model retrieves relevant context
- Answer Accuracy: how accurately the model answers questions based on the context

6 Conclusion

In conclusion, the Retrieval Augmentation Generation (RAG) framework is a powerful tool that enables LLMs to incorporate real-time information, giving accurate responses to user queries. We were able to explore a Retrieval-Augmented Generation (RAG) system tailored to the domain of human nutrition. However, it is also important to keep in mind that the traditional RAG may not yield optimal results in complex scenarios and therefore, enhancing RAG performance for the user by enhancing the retrieval accuracy of the RAG pipeline is advisable.

7 Future Scope

The next part of this project includes enhancing the privacy of the current RAG pipeline. This would involve the following steps.

- Choose a suitable LLM model to generate an output based on the context of the relevant chunks retrieved in this stage.
- Perform exploratory data analysis to justify the choice of our LLM model.
- Load the chosen LLM locally
- Generate text with the locally loaded LLM model
- Apply prompt engineering techniques for Augmentation to complete the RAG pipeline.

References