Member-only story

Improve RAG Pipelines With These 3 Indexing Methods

The data we retrieve doesn't have to be the same as the data we used while indexing

Ahmed Besbes · Follow

Published in Level Up Coding

4 min read · Nov 10

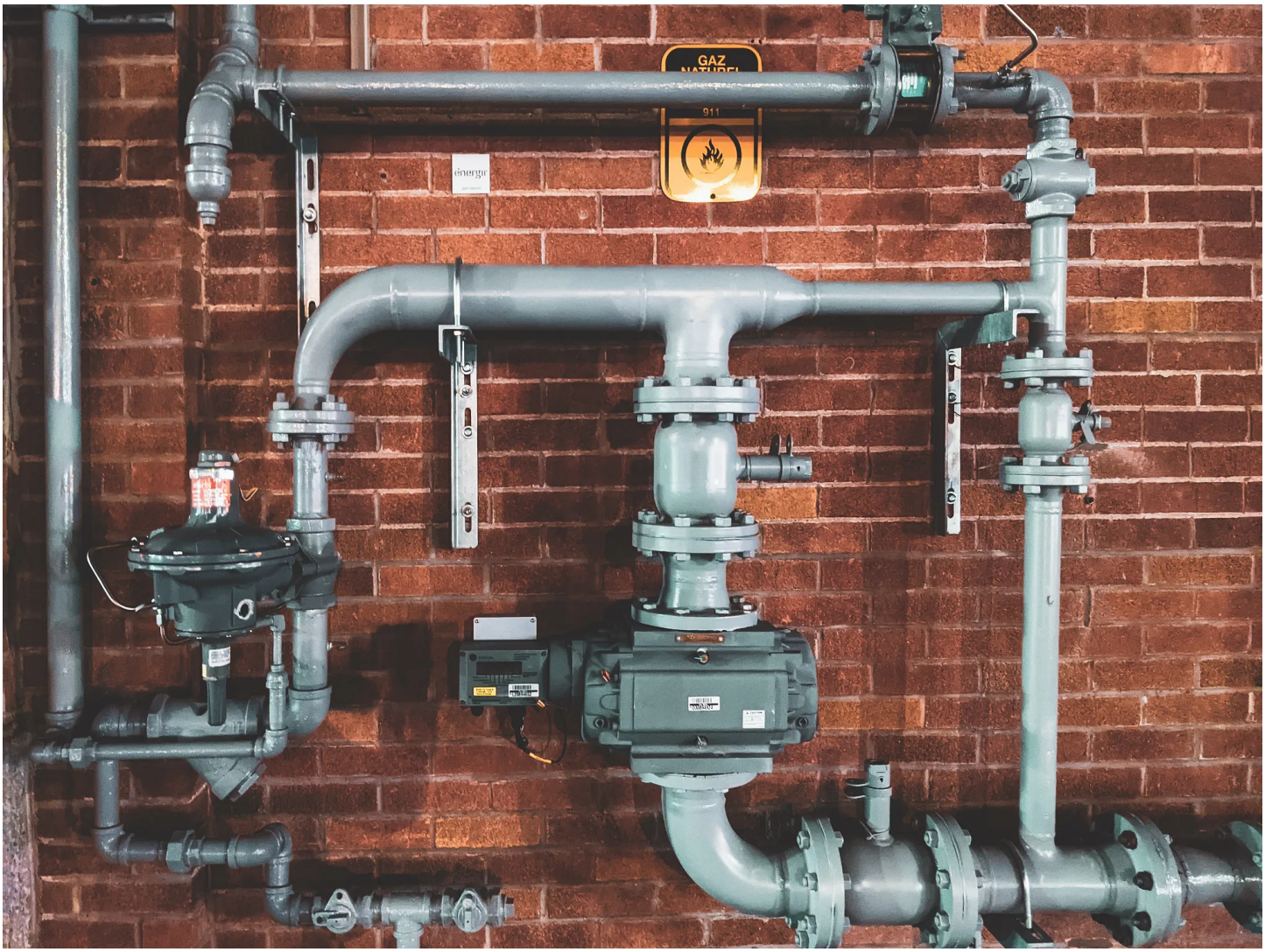

Photo by Sigmund on Unsplash

The Retrieval Augmented Generation framework, or simply RAG, has seen increasing popularity lately as a practical application of LLMs.

Thanks to the plethora of frameworks, it became extremely easy to prototype this system to build applications that serve different purposes: chatting with PDFs, generating SQL code from English instructions, building assisted documentation, etc.

Prototyping RAGs is easy. It's only a matter of a few lines of code.

Building production-ready RAGs, however, is hard. This requires careful tuning and a lot of engineering to make such a system consistent and reliable.

> **In this article, I discuss data indexing when building RAG.**
>
> **I'll go beyond the common indexing method we typically use and explore 3 different ways you can index your data.**

These methods are worth experimenting with as they boost the **performance** and **accuracy** of your applications.

Let's see how.

> *If you're interested in practical tips to increase your productivity in building ML systems, you feel free to subscribe to my newsletter: The Tech Buffet.*
> *I send weekly insights in programming and system design to help you ship AI products faster.*

Reminder on data indexing in typical RAGs

In the default implementation of a RAG system, documents are first split into chunks.

Then, each chunk is embedded and indexed into a vector database.

In the retrieval step, the input query is also embedded and the most similar chunks are extracted.

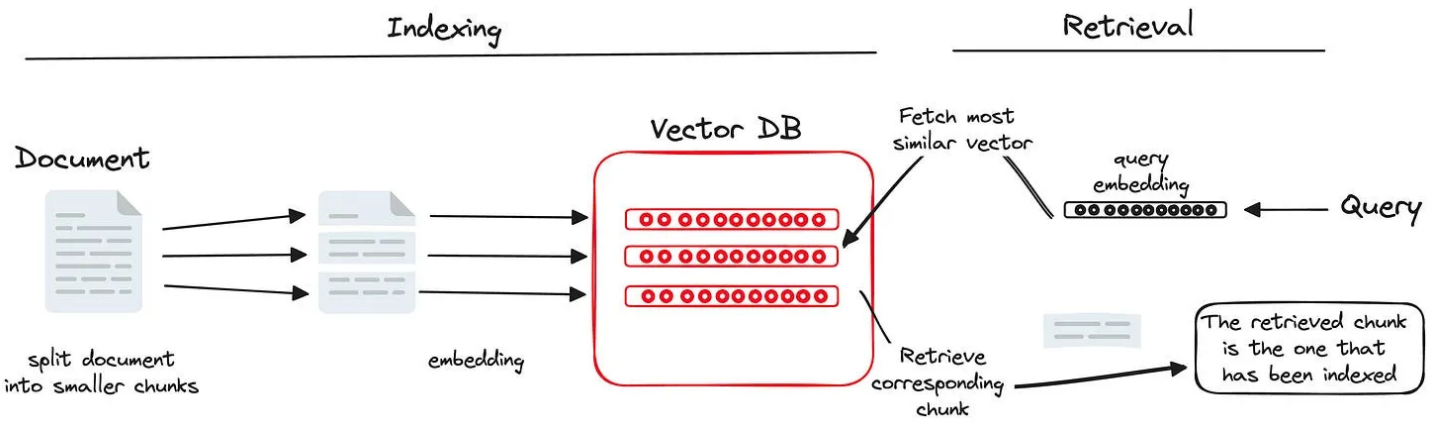

Image by the author

In this setup, the data (i.e. *the chunks*) we retrieve is the same as the data we index.

This is the most natural and intuitive implementation.

🚨 **However, this doesn't have to be always the case.**

We can index the chunks differently to increase the performance of such systems

the data we retrieve doesn't have to be the same as the data we used while indexing.

1 — Index chunks by their subparts 🧩

Instead of indexing the whole chunk directly, we can split it again into smaller pieces (e.g. sentences) and index it with those multiple times.

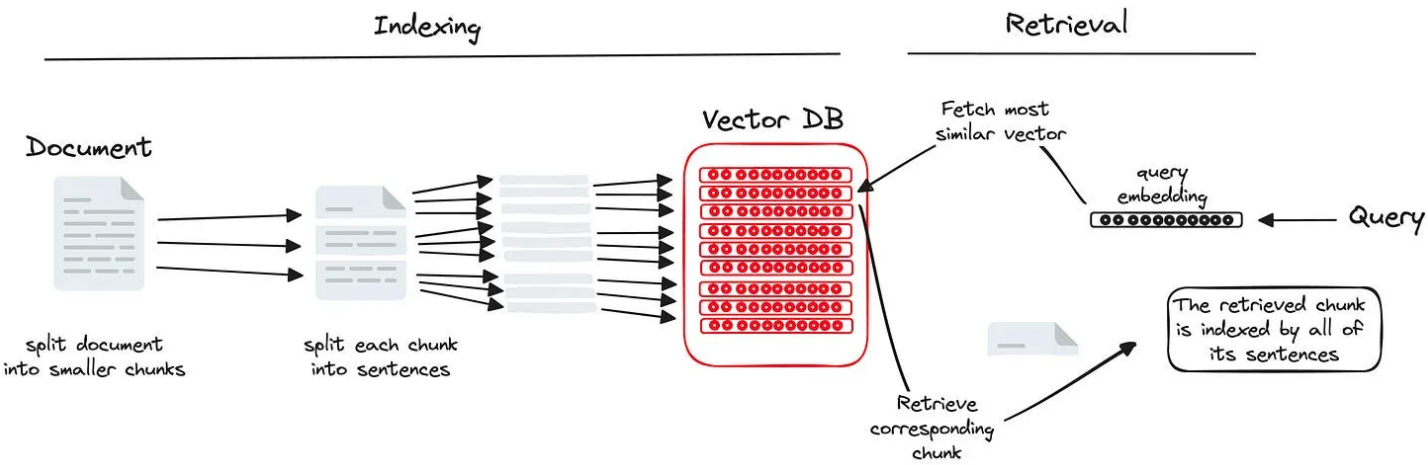

Image by the author

> **Why is this useful?**

Imagine dealing with long and complex chunks that discuss multiple topics or conflicting information. Using them in a typical RAG will likely generate noisy outputs with some irrelevant content.

If we separate these chunks into smaller sentences, each sentence will likely have a clear and well-defined topic that matches the user query more accurately.

When retrieving the chunk, this method makes sure to get a relevant context to the query and a broader context (not present in the indexing sentence) that will be useful for the LLM to generate a comprehensive answer.

2 — Index chunks by the questions ❓ they answer

Instead of indexing the chunks directly, we can instruct the LLM to generate the questions they answer and use them for indexing. This is a simple approach.

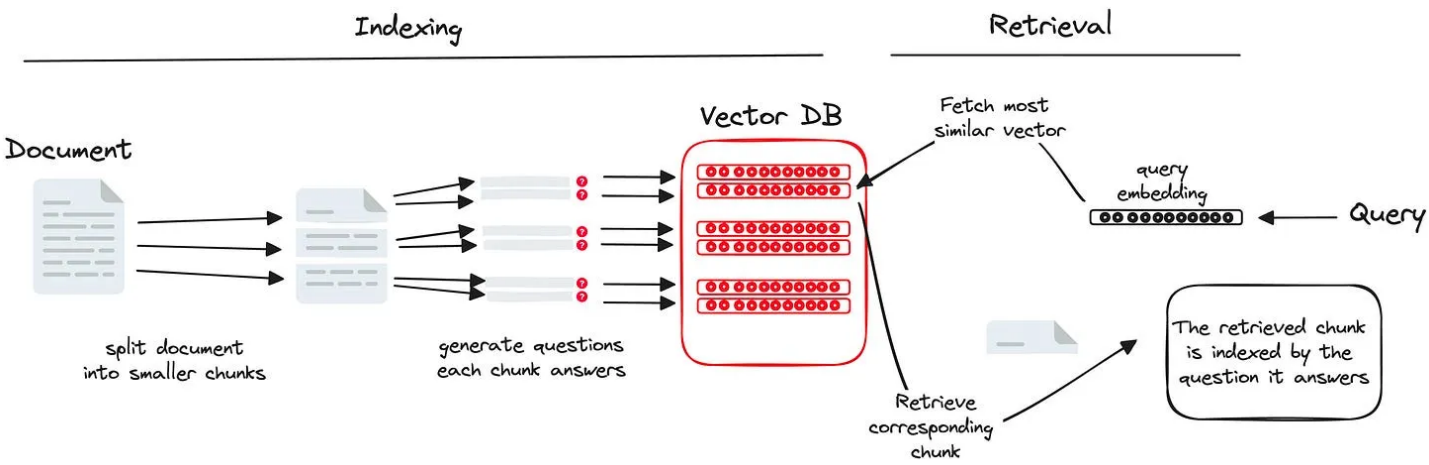

Image by the author

When submitting the query, the RAG will therefore compare it to the most relevant questions that the data answers. Then, based on these questions, it will retrieve the corresponding chunks.

This indexing method is useful because it aligns the user's query/objective with the core content of the data.

If a user doesn't formulate a very clear question, such an indexing method can reduce ambiguity. Instead of trying to figure out what chunks are relevant to the

user's question, we directly map it to existing questions that we know we have an answer for.

3 — Index chunks by their summaries 📝

This indexing method is similar to the previous one. It uses the chunk summary for indexing instead of the questions it answers.

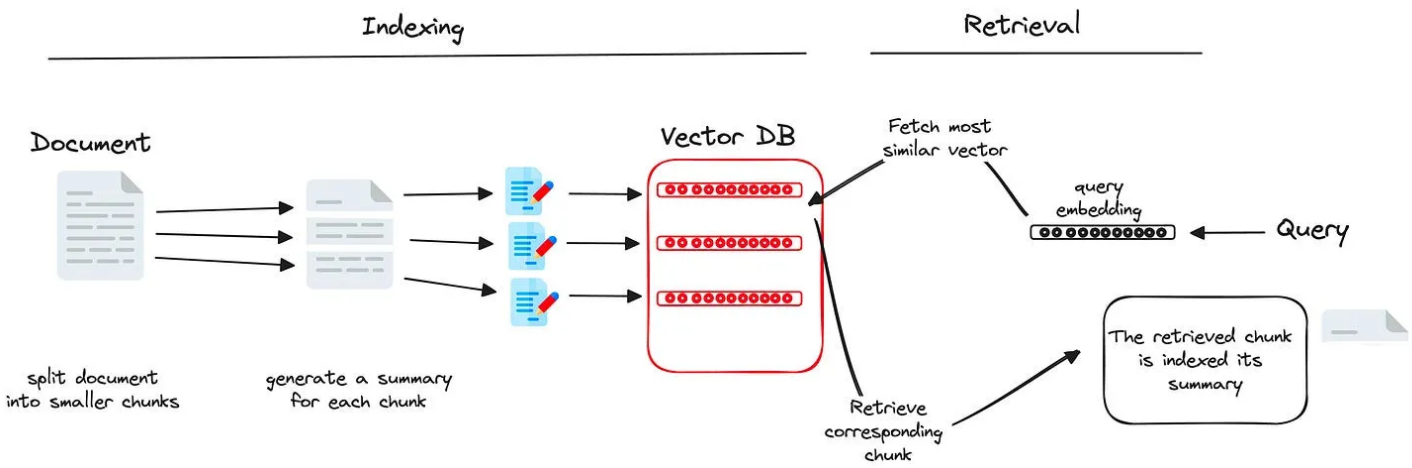

Image by the author

This is typically useful when the chunks have redundant information or irrelevant details that are not useful to the user's query.

It also captures the essence of the information that the user is looking for.

Conclusion

I hope you've enjoyed this article and learned something useful about improving RAG indexing.

If you know about other indexing methods that improve RAG systems, please tell me about them in the comments.

Interested in building RAGs for your company? Have a look at my next post where I cover a curated list of top articles from the industry.

Until next time! 👋

Machine Learning | Artificial Intelligence | Data Science | Programming