

Open in app ↗



Search

Write



Building LLM Applications: Introduction (Part 1)

Vipra Singh · [Follow](#)

5 min read · Jan 8, 2024



905



3



Learn Large Language Models (LLM) through the lens of a Retrieval Augmented Generation (RAG) Application.

Posts in this series

1. [Introduction](#) (This Post)
2. [Data Preparation](#)
3. [Sentence Transformers](#)
4. [Vector Database](#)
5. [Search & Retrieval](#)
6. [LLM](#)
7. [Open-Source RAG](#)

8. Evaluation

9. Serving LLMs

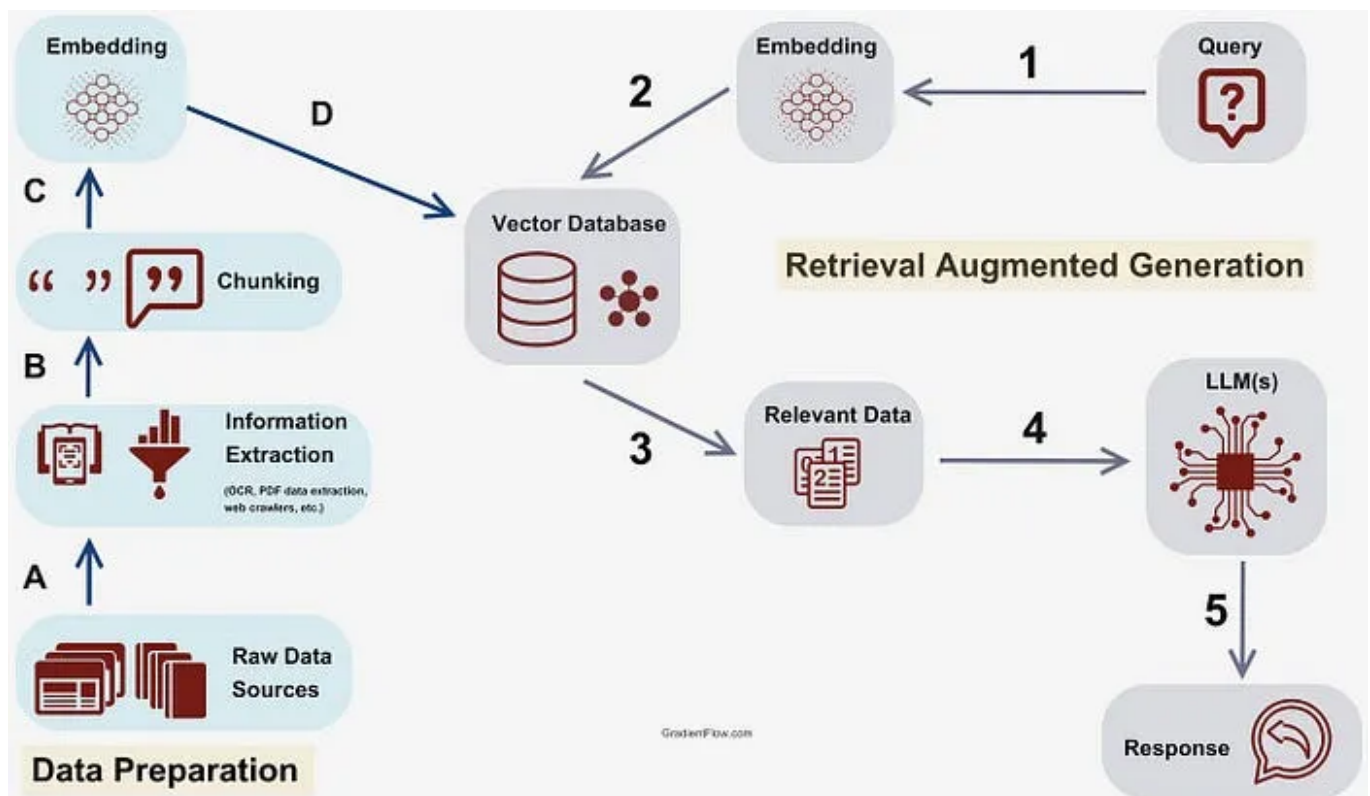
10. Advanced RAG

. . .

Table of Contents

- 1. What is Retrieval Augmented Generation?
- 2. Why RAG?
- 3. High-Level RAG Architecture
- 4. Conclusion
- Credits

. . .



Even a simple RAG application entails tuning many different parameters, components, and models. | Credits:
Ben Lorica

Greetings!

In my recent exploration of Language Model (LLM) applications, I've been captivated by the significant role of Retrieval Augmented Generation (RAG). Understanding the end-to-end RAG architecture, from its conceptualization to its deployment over the cloud, can be quite challenging.

To address this, I'm thrilled to announce a forthcoming series of detailed blogs where I will unravel the intricacies of LLM through the lens of RAG application, providing a comprehensive understanding of each stage of the RAG pipeline along with practical, hands-on experience.

I aim to make the complex world of LLM more accessible to everyone, ensuring that each stage is covered in detail.

Join me on this enlightening journey as we delve into the potential of LLM applications through the comprehensive study of RAG.

1. What is Retrieval Augmented Generation?

If you have been looking up data in a vector store or some other database and passing relevant info to LLM as context when generating output, you are already doing retrieval augmented generation. Retrieval augmented generation or RAG for short is the architecture popularized by Meta in 2020 that aims to improve the performance of LLMs by passing relevant information to the model along with the question/task details.

2. Why RAG?

LLMs are trained on large corpora of data and can answer any questions or complete tasks using their parameterized memory. These models have knowledge cutoff dates depending on when they were last trained. When asked a question out of its knowledge base or about events that happened after the knowledge cutoff date, there is a high chance that the model will hallucinate. Researchers at Meta discovered that by providing relevant information about the task at hand, the model's performance at completing the task improves significantly.

For example, if the model is being asked about an event that happened after the cutoff date, providing information about this event as context and then asking the question will help the model answer the question correctly. Because of the limited context window length of LLMs, we can only pass the most relevant knowledge for the task at hand. The quality of the data we add in the context influences the quality of the response that the model generates. There are multiple techniques that ML practitioners use in different stages of an RAG pipeline to improve LLM's performance.

3. High-Level RAG Architecture

LangChain has an example of RAG in its smallest (but not simplest) form:

A typical RAG application has two main components:

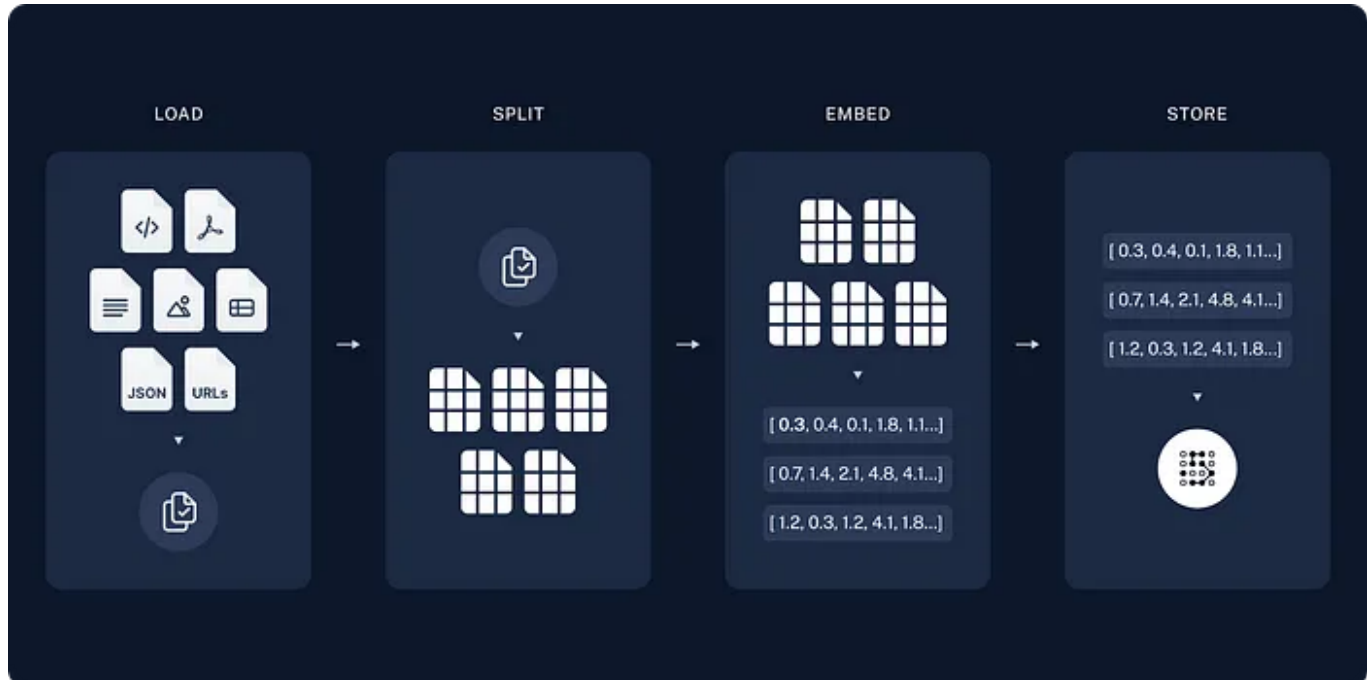
Indexing: A pipeline for ingesting data from a source and indexing it. This usually happens offline.

Retrieval and Generation: The actual RAG chain, which takes the user query at run time and retrieves the relevant data from the index, then passes that to the model.

The most common full sequence from raw data to answer looks like this:

Indexing

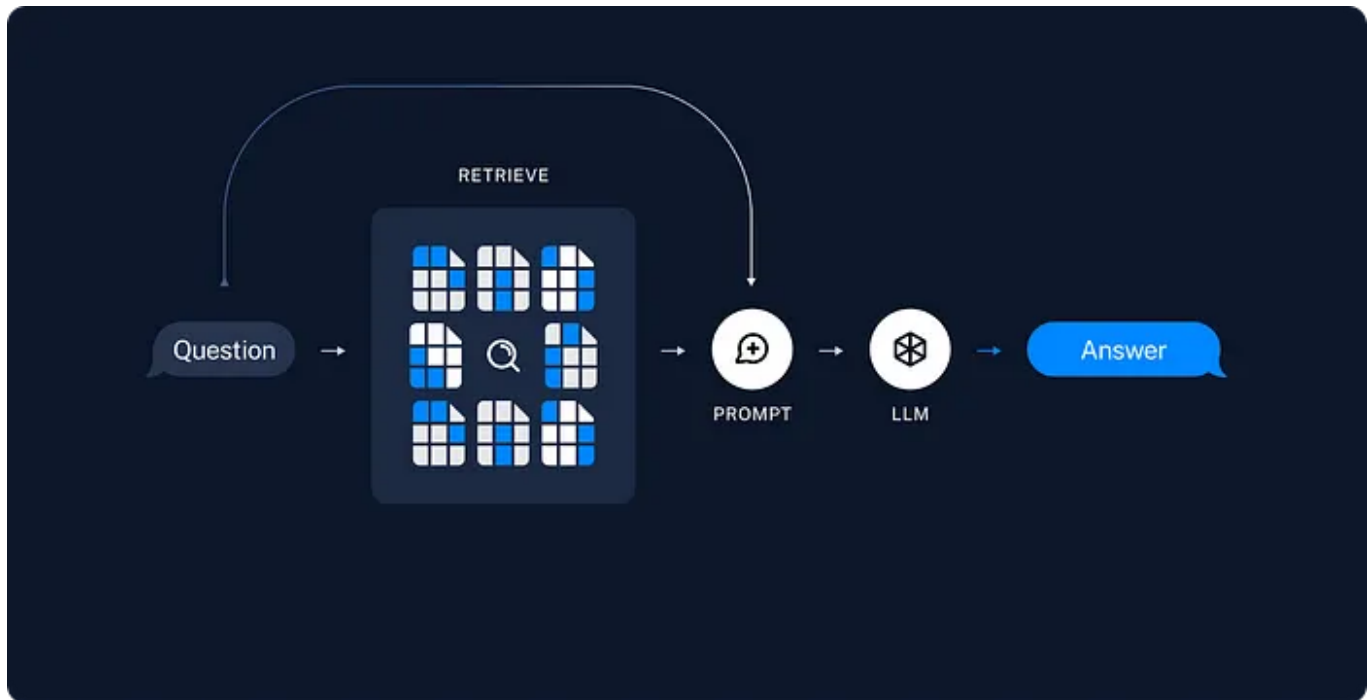
1. **Load:** First we need to load our data. This is done with DocumentLoaders.
2. **Split:** Text splitters break large Documents into smaller chunks. This is useful both for indexing data and for passing it into a model, since large chunks are harder to search over and won't fit in a model's finite context window.
3. **Store:** We need somewhere to store and index our splits so that they can later be searched. This is often done using a VectorStore and Embeddings model.



Credits: [Langchain](#)

Retrieval: Given a user input, relevant splits are retrieved from storage using a Retriever.

Generation: A ChatModel / LLM produces an answer using a prompt that includes the question and the retrieved data



Credits: [Langchain](#)

```
from langchain.document_loaders import WebBaseLoader
from langchain.indexes import VectorstoreIndexCreator
loader = WebBaseLoader("https://www.promptingguide.ai/techniques/rag")
index = VectorstoreIndexCreator().from_loaders([loader])
index.query("What is RAG?")
```

With these five lines, we get a description of RAG, but the code is heavily abstracted, so it's difficult to understand what's happening: We fetch the contents of a web page (our knowledge base for this example).

1. We process the source contents and store them in a knowledge base (in this case, a vector database).

2. We input a prompt, LangChain finds bits of information from the knowledge base and passes both prompt and knowledge base results to the LLM.

While this script is helpful for prototyping and understanding the main beats of using RAG, it's not all that useful for moving beyond that stage because you don't have much control. Let's discuss what goes into implementation.

4. Conclusion

In this first part of the LLM Apps series, we dissected the vital role of Retrieval Augmented Generation (RAG) in enhancing Large Language Models (LLMs). From understanding the motivation behind RAG to exploring the components of an LLM application architecture, we unveiled the layers that make these applications tick.

We delved into the intricacies of knowledge base retrieval, emphasizing the importance of ETL pipelines and tools like Unstructured, LlamaIndex, and LangChain's Document loaders. The significance of maintaining an updated knowledge base was highlighted, with a nod to efficient document indexing processes.

Join us in the upcoming blogs as we continue our journey through the remaining stages of the RAG pipeline. From User Query Processing to Front-End Development, we'll provide hands-on insights to demystify LLM applications. Together, let's unlock the full potential of LLMs, making the complex accessible to all in the realm of natural language understanding and generation. Stay tuned for more!

Credits

In this blog post, we have compiled information from various sources, including research papers, technical blogs, official documentations, YouTube videos, and more. Each source has been appropriately credited beneath the corresponding images, with source links provided.

Below is a consolidated list of references:

1. <https://llmstack.ai/blog/retrieval-augmented-generation>
2. https://python.langchain.com/v0.1/docs/use_cases/question_answering/#quickstart
3. https://python.langchain.com/v0.1/docs/use_cases/question_answering/#quickstart

Thank you for reading!

If this guide has enhanced your understanding of Python and Machine Learning:

- Please show your support with a clap 🖐️ or several claps!
- Your claps help me create more valuable content for our vibrant Python or ML community.
- Feel free to share this guide with fellow Python or AI / ML enthusiasts.
- Your feedback is invaluable — it inspires and guides my future posts.

Connect with me!

[Vipra](#)

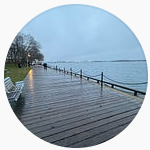
Large Language Models

Retrieval Augmented

Semantic Search

Langchain

Embedding



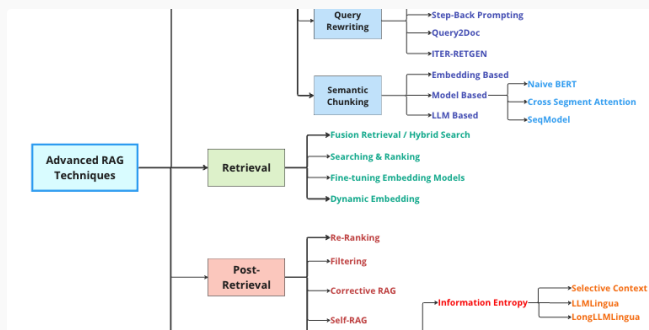
Written by Vipra Singh

1K Followers

Follow



More from Vipra Singh

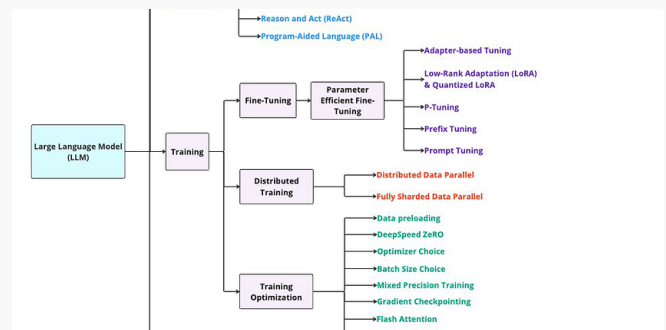


Vipra Singh

Building LLM Applications: Advanced RAG (Part 10)

Learn Large Language Models (LLM) through the lens of a Retrieval Augmented...

🌟 · 48 min read · Apr 27, 2024



Vipra Singh

Building LLM Applications: Large Language Models (Part 6)

Learn Large Language Models (LLM) through the lens of a Retrieval Augmented...

38 min read · Feb 10, 2024



384



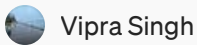
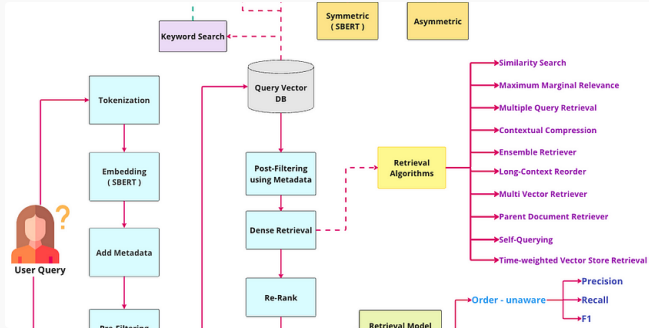
2



390



1



Vipra Singh

Building LLM Applications: Search & Retrieval (Part 5)

Learn Large Language Models (LLM) through the lens of a Retrieval Augmented...

26 min read · Jan 27, 2024



405



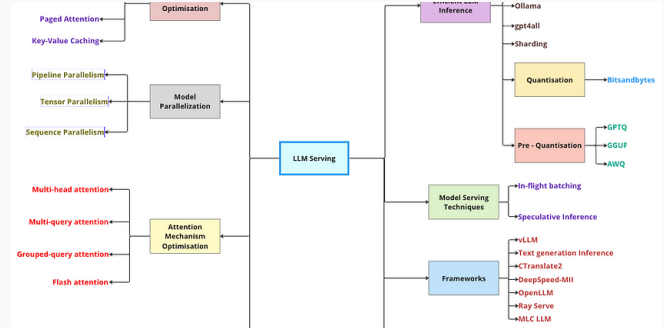
1



546



3



Vipra Singh

Building LLM Applications: Serving LLMs (Part 9)


Learn Large Language Models (LLM) through the lens of a Retrieval Augmented...

🌟 · 50 min read · Apr 17, 2024

See all from Vipra Singh

Recommended from Medium



 Paul Iusztin in Decoding ML

The 4 Advanced RAG Algorithms You Must Know to Implement

Implement from scratch 4 advanced RAG methods to optimize your retrieval and post-...

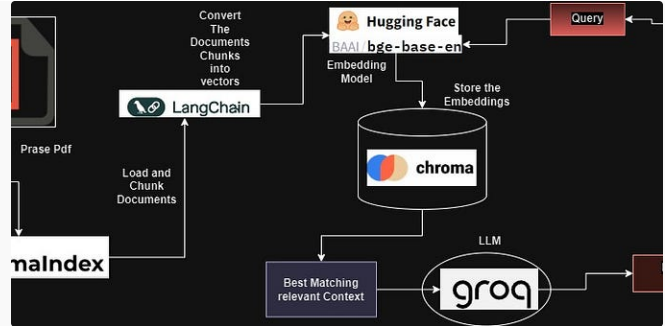
15 min read · May 4, 2024




1.4K



10



 Plaban Nayak in The AI Forum

RAG on Complex PDF using LlamaParse, Langchain and Groq

Retrieval-Augmented Generation (RAG) is a new approach that leverages Large Language...

13 min read · Apr 7, 2024



678



9



Lists



Natural Language Processing

1494 stories · 1011 saves



Generative AI Recommended Reading

52 stories · 1106 saves



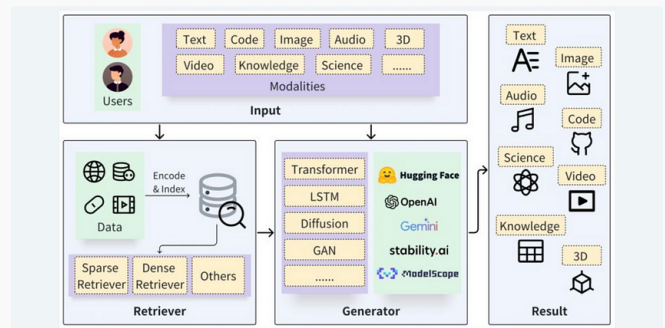
AI Regulation

6 stories · 473 saves



ChatGPT prompts

47 stories · 1642 saves





Fareed Khan in Level Up Coding

Building LLaMA 3 From Scratch with Python

Code Your Own Billion Parameter LLM

29 min read · May 28, 2024



1.1K



11



Pavan Belagatti in Generative AI

Implementing Retrieval Augmented Generation (RAG): A...

Large language models (LLMs) are becoming the backbone of most of the organizations...

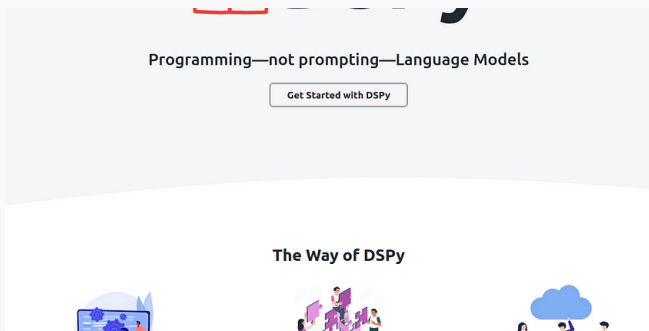
7 min read · Apr 17, 2024



287



1



Vishal Rajput in AIGuys

Prompt Engineering Is Dead: DSPy Is New Paradigm For Prompting

DSPy Paradigm: Let's program—not prompt—LLMs



· 11 min read · May 29, 2024



1K



14



Anton Antich in Superstring Theory

Retrieval Augmented Generation (RAG) In 2 Minutes

What's the fuzz all about and how to build my custom agent or "chatbot" easy and fast

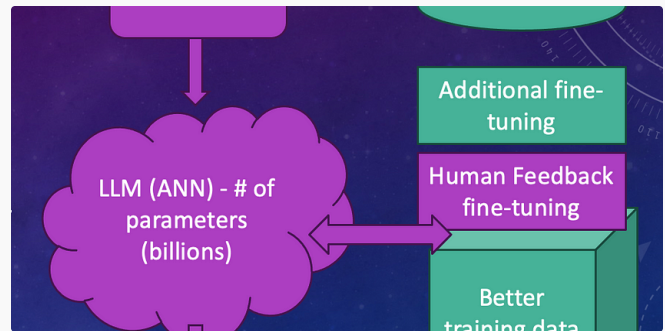
4 min read · May 2, 2024



35



1

[See more recommendations](#)