Retrieval-Augmented Generation with LlamaIndex

https://github.com/danielbank/rag-llamaindex

Daniel Bank

A Brief Review

What is Generative AI?

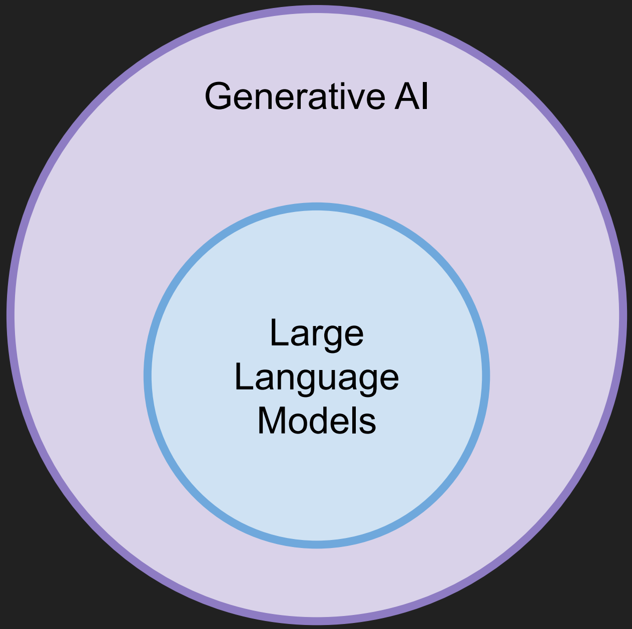
AI that can produce various types of content including text, images, audio, and data.

Examples:

- RNNs (Recurrent Neural Networks / LSTM (Long Short-term Memory)
- GAN (Generative Adversarial Networks) [*generator and discriminator networks*]
- Transformer-based Models (BERT, GPT, etc)

What is an LLM?

Large Language Models (LLM) are machine learning models that can comprehend and generate human language text.



Examples:

- GPT-4 (OpenAI) [*ChatGPT is the chat bot*]
- Gemini (Google)
- Claude 3 (Anthropic)

What are the components of a prompt?

- **Instructions**: a specific task you want the model to perform.
- **Constraints**: restrictions on acceptable responses.
- **Input Data**: the data that we are interested to find a response for.
- **Output Indicator**: the type or format of the output.
- **Examples:** demonstrations that steer the model to better performance.
- **Context**: external information that shapes the response.

How do we teach an LLM
about our data
(data it wasn’t trained on)
?

We could train the LLM on our data.

As seen in the demo from the Getting Started with Generative AI APIs presented at the February 2024 Phoenix AI Devs Meetup.

Pros

- Consistent Output Quality
- Less Dependent on External Data

Cons

- Expensive to Train
- Static Knowledge

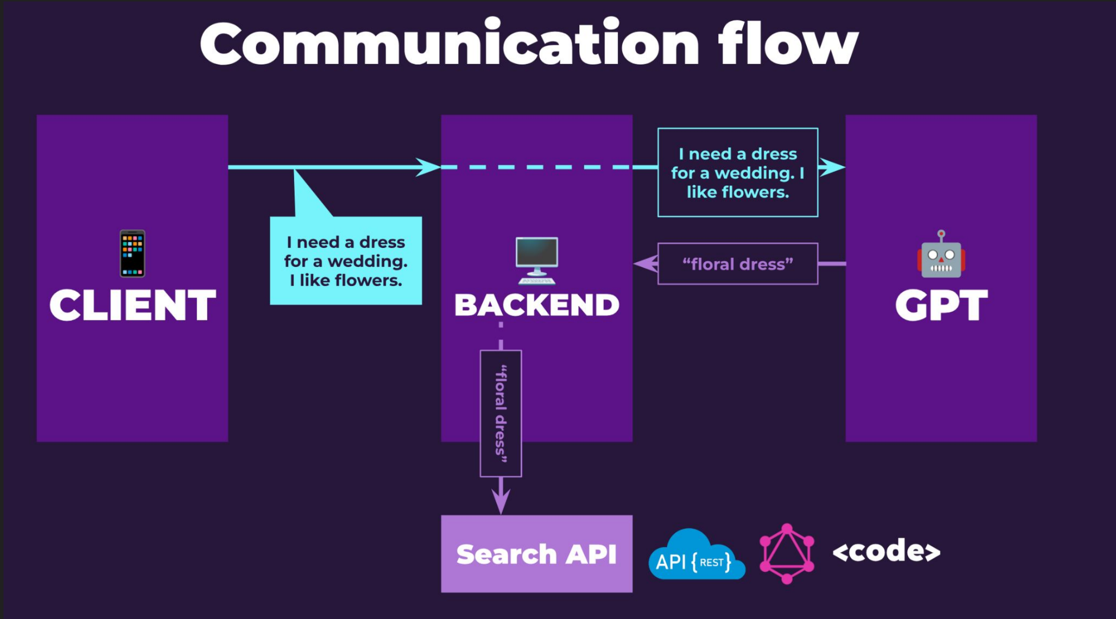
We could leverage a tool outside the LLM.

As seen in Ben Ilegbodu's GPT-Powered AI Shopping presentation from HalfStack Phoenix. Here, the LLM translates unstructured input into structured input and then performs a server call with it.

NOTE:

LLM is not the "Frontend"

Backend acts as the intermediary



Open AI function calling

We could put include our data as context in the prompt (while still leveraging a tool outside the LLM).

- Original Paper: [Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks (2021)]

- Traditional RAG Pipeline Steps:
  1. Retrieve some context from the DB
  2. Stick that context into the LLM Prompt
  3. Call the LLM a single time to get a response

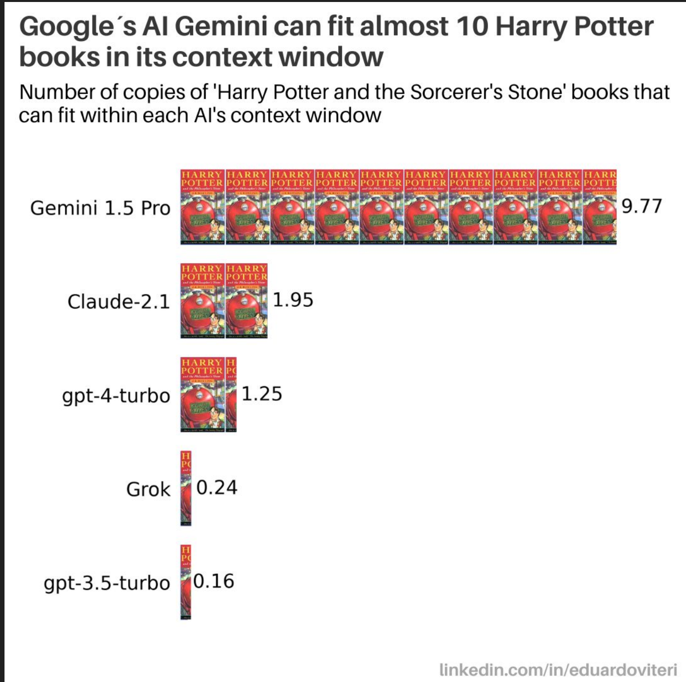
- DeepLearning.ai Course: [JavaScript RAG Web Apps with LlamaIndex]

We could use Agentic RAG (outside the scope of this talk).

- Standard RAG Pipeline
  - Good for simpler questions over a small set of documents
- Agentic RAG Pipeline
  - Good for multiple step processing with possible dependencies of steps on each other
- How it works:
  - **Routing**: Add decision-making to route requests to multiple tools
  - **Tool Use**: Create an interface for agents to select a tool and provide the right arguments
  - **Multi-step reasoning with tool use**: Use an LLM to perform multi-step reasoning with a range of tools for retaining memory throughout that process
- DeepLearning.ai Course: Building Agentic RAG with LlamaIndex

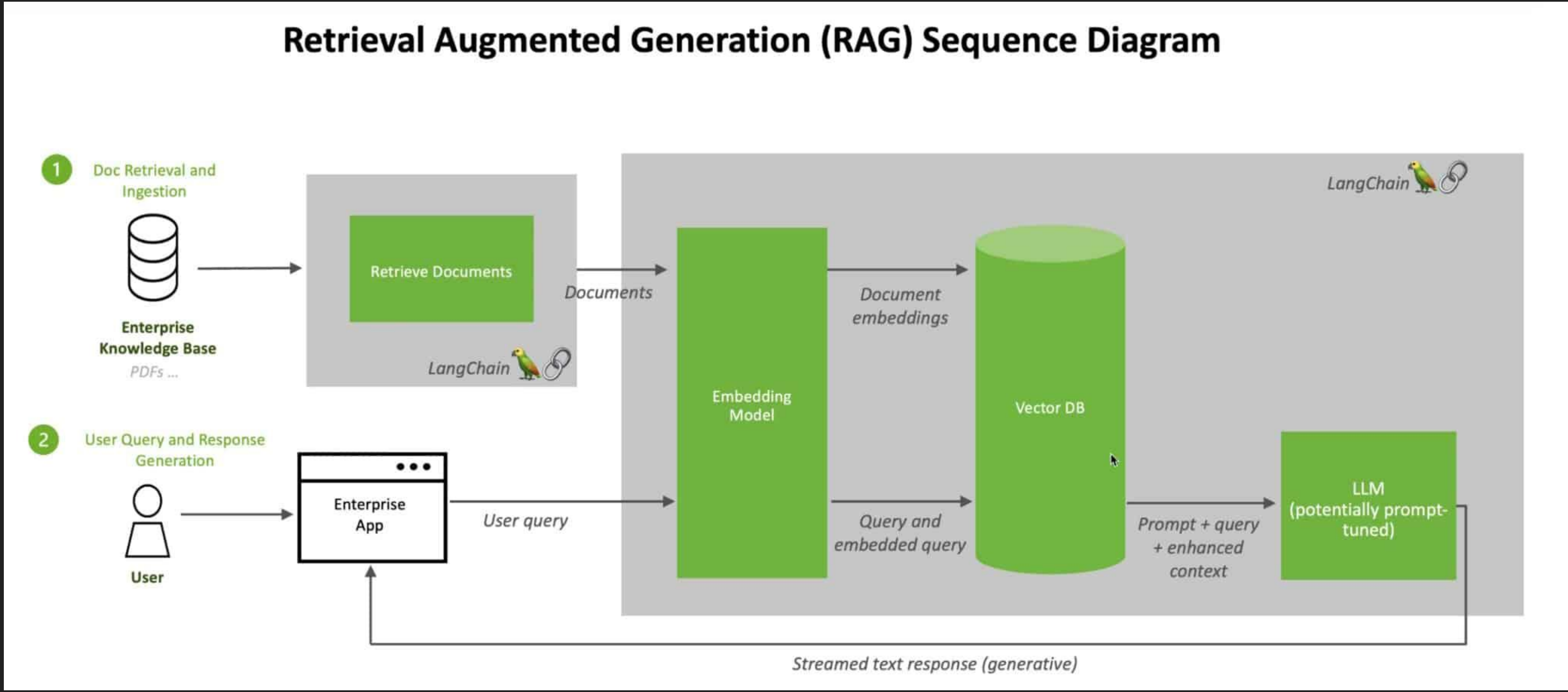
We could put our data as context in the prompt?

- Traditional RAG, you need to filter down the context you care about
- With a larger context window, you don't have to be as precise or prescriptive.



Let’s Dive Deeper on RAG

Traditional RAG Pipeline (from NVIDIA's blog)

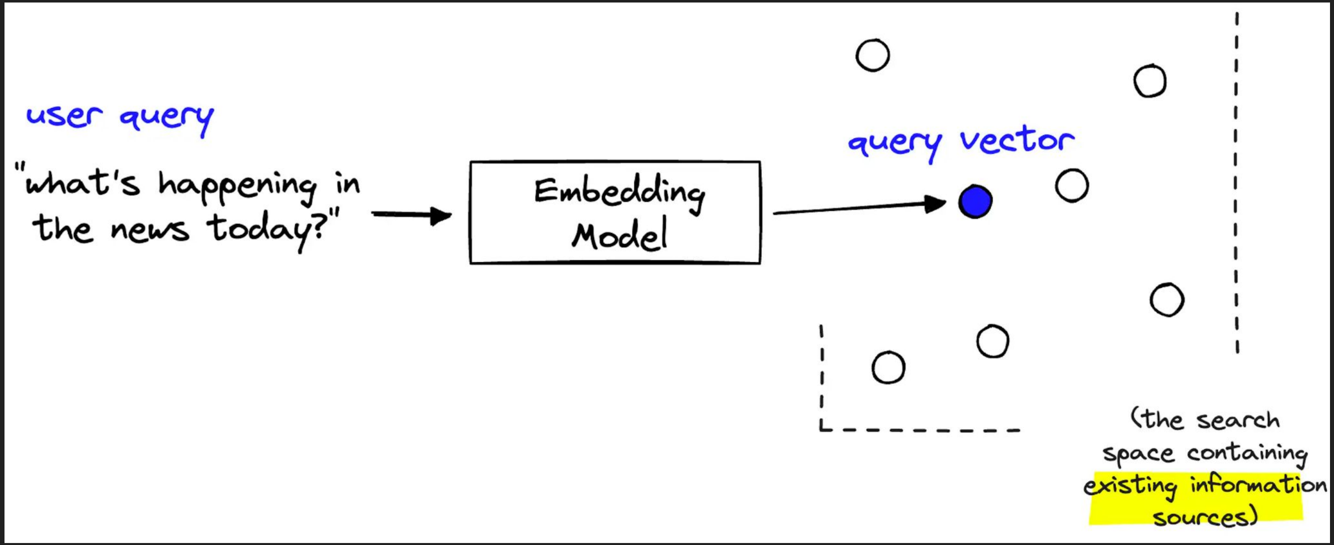


Retrieving Documents

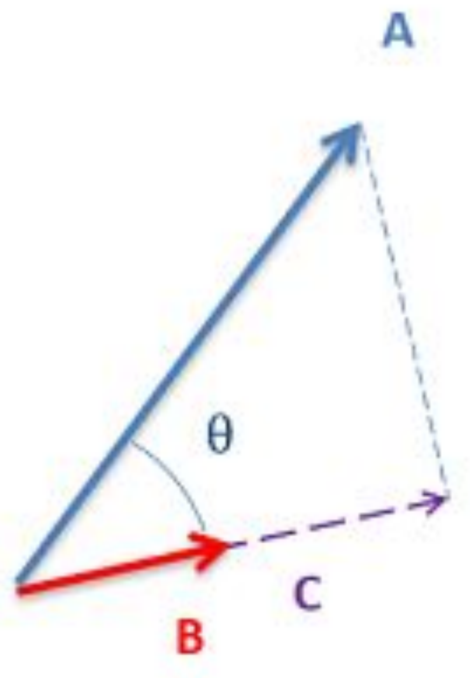
- Get the data we intend to insert into the Vector DB in a well-defined shape
  - Example: We have text data in different file types (.md, .pdf, .csv, etc.)
- LlamaParse is a proprietary parsing service, exceptionally adept at transforming PDFs containing complex tables into a neatly organized markdown format:
  - What language to use for OCR?
  - Is there any text at different orientations?
  - Are there special instructions for the Embedding Model?

Embedding Models

- An embedding model is a neural network that converts text into dense vector representations (embeddings).
- Different types of Embedding Models based on modality (text, image, audio)
- [Massive Text Embedding Benchmark (MTEB)]



Vector Similarity (Dot Product)



$$\vec{A} \cdot \hat{B} = |A||B| \cos(\theta)$$

if the magnitude of B is 1, then...

$$C = \vec{A} \cdot \hat{B} = |A| \cos(\theta)$$

Databases (A gentle introduction to Vector DBs)

- Relational Databases
    - When we think of databases, relational databases are usually what we picture
    - Great for Structured Data
    - Data is search by columns

ISBN	Year	Name	Author
0767908171	2003	A Short History of Nearly Everything	Bill Bryson
039516611X	1962	Silent Spring	Rachel Carson
0374332657	1998	Holes	Louis Sachar

Vector Databases (A gentle introduction to Vector DBs)

- Vast majority of data on the internet is unstructured (images, text, audio, video)
- Data is searched via content rather than keywords (similarity score, e.g. dot product)
  - Side note: PageRank (1998), the algorithm that powers Google Search, uses eigenvectors to determine page rank (similarity of pages to the query keywords)

Data UID[1]	Vector representation
00000000	[-0.31, 0.53, -0.18, ..., -0.16, -0.38]
00000001	[0.58, 0.25, 0.61, ..., -0.03, -0.31]
00000002	[-0.07, -0.53, -0.02, ..., -0.61, 0.59]

Let’s build RAG with Milvus (a Vector DB) in Google Colab

Milvus Bootcamp - Build RAG with Milvus Tutorial

- Milvus Bootcamp is a good resource to learn about the Milvus Vector DB: https://github.com/milvus-io/bootcamp

- Specifically, there is a Jupyter Notebook Tutorial for Building RAG with Milvus that we can run in a Google Colab: https://github.com/milvus-io/bootcamp/blob/master/bootcamp/tutorials/quickstart/build_RAG_with_milvus.ipynb

Colab Tweeks

- Colab specifies a higher version of grpcio

```
! pip uninstall -y grpcio
! pip install grpcio==1.63.0
! pip install --upgrade pymilvus openai requests tqdm
```

- Don’t put your **OPENAI_API_KEY** in the Notebook contents

```
import os
from google.colab import userdata
os.environ["OPENAI_API_KEY"] = userdata.get('OPENAI_API_KEY')
```

Let's build a
RAG-Powered Web App
with LlamaIndex and Next.js

Demo Time

- GitHub Repo: https://github.com/danielbank/rag-llamaindex

LlamaIndex - https://github.com/run-llama/LlamaIndexTS

- Create LlamaIndex App is a CLI Tool for Bootstrapping RAG Web Apps
- `npx create-llama@latest` - Run the CLI tool to bootstrap a web app
- `npm run generate` - Create the Document Embeddings for the knowledge-base data in `./data`
- `npm run dev` - Run the web app

The Shape of the Data Matters

- Depending on the data, you may need to do pre-processing or use a beefier Embedding Model or more dimensions

- `BadRequestError: 400 This model's maximum context length is 8192 tokens, however you requested 77935 tokens (77935 in your prompt; 0 for the completion). Please reduce your prompt; or completion length.`

The Kind of Data Matters (only text)

- The parser only supports text data in the quickstart example. If your ./data folder has any images, you will get an error

- `Error: Cannot calculate image nodes embedding without 'imageEmbedModel' set`

Demo Time

- Let’s replace the silly example with the codebase for Histogramo, an Android App for detecting dice rolls