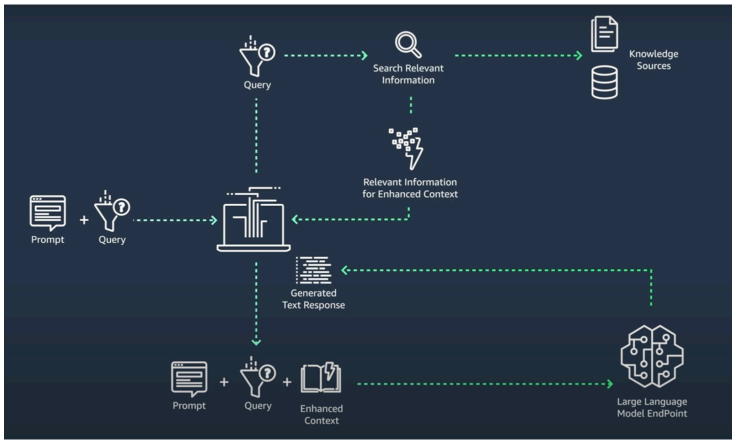
RAG-implementation Outline

Overview of RAG (Retrieval Augmented Generation)

- Retrieval Augmented Generation (RAG) enhances generative AI models by integrating external knowledge sources to ensure contextually relevant responses.
- RAG improves generative AI efficiency and accuracy by accessing up-to-date information from external sources, providing more precise and relevant outputs.


Query
Search Relevant Information
Knowledge Sources
Relevant Information for Enhanced Context
Prompt
+
Query
Generated Text Response
Prompt
+
Query
+
Enhanced Context
Large Language Model EndPoint

How RAG Works

- Process flow:

- Prompt used to retrieve contextualized information from a database: A prompt is issued to search the database for information that is contextually relevant to the query.
- Feeding contextualized information and prompt to an LLM: The retrieved contextualized information, along with the original prompt, is then fed into a large language model (LLM) to generate a coherent and accurate response.

-Retrieving contextualized information from a database:

- Role of Vectorized Databases in Capturing Semantic Relationships: Vectorized databases capture semantic relationships by representing the meanings of the text using embeddings. This allows the LLM to associate them with the correct contextualized information from the database.

Important tools

<u>-RAG frameworks:</u>

LangChain

a) Multiple LLM Interfaces: LangChain supports interfaces for OpenAI, HuggingFace, and PaLM.
b) End-to-End Application Support: It is better for end-to-end applications as it is a more complete framework.
c) Intuitive Interface for MVP: LangChain offers a more intuitive interface, which can be beneficial for MVP development.
d) Vectorizes Databases: LangChain provides a set of abstractions for converting the database into a vector set, memory, etc., making it simpler but less flexible.
e) Implementation: LangChain is written in Python; for C++, you can use FAISS, Annoy, or Usearch.

LlamaIndex

a) Advanced RAG for LLM and Data Connectors: LlamaIndex is more used for advanced RAG using LLM and data connectors.
b) Robust for Data Querying and Indexing: It is focused on querying data, making it robust for storage and indexing.
c) Flexible Memory Building: LlamaIndex offers more flexibility with building memory, allowing deeper LLM to data connections.

-Text splitters:

Choose a text splitter to split data into chunks - You want to maintain semantic relations between chunks

a) Recursive splitter - recommended as it keeps related sections of text together
b) character/token - splits text based on specific desired characters/tokens
c) Semantic - splits based on embeddings of sentences alone

-Vector databases:

a) FAISS: FAISS is used for nearest neighbor searching and is best for large-scale applications.

b) ChromaDB: ChromaDB offers nearest neighbor searching and is scalable with payment options. An in-memory RAG example is Chroma, an in-memory RAG for learning.

c) Elastic Search: has slow indexing and data encryption.

Implementation Steps

a) Identify Key Points of User Prompt: Begin by dissecting the user prompt to pinpoint its crucial elements, such as stemming and embedding requirements. This step lays the foundation for understanding the context and requirements of the task.
b) Find Appropriate Database to Simulate User Information: Next, select or create a suitable database containing user information relevant to the simulated scenario. This database will serve as the repository from which data chunks will be retrieved.
c) Search for Data Chunks Relevant to Prompt Using Embeddings and Search Algorithms: Employ sentence embeddings to convert textual data into vector representations, facilitating efficient search operations. Implement robust search algorithms to sift through the database and extract the most pertinent data chunks corresponding to the identified key points.
d) Retrieve the Data Chunks and Feed Them into an LLM: Once the relevant data chunks are identified, feed them into a Language Model (LM), leveraging its capabilities to generate responses or insights. The LM processes the input data and produces coherent output aligned with the user prompt and the context of the simulated scenario.

Advantages and Disadvantages

Pros:

- Creating personal rag will allow for privacy of personal information (for example a salesman user)
- Allows for customization (guiding the user to certain dataset endpoints based on prompts)

Cons:

- Openai updates to chatgpt make uploading files easy and accessible
- Openai’s current Assistant + file api is seamless, although not as much for large volumes of data as it is limited to 20 files

Enhancements

a) Semantic router:
   -A super fast decision making layer for the LLM
   1) Allows user to configure deterministic responses to specific triggers
   2) Eg -> one input prompt can trigger a different set of rules for an agent

b) Semantic kernel:
   - Microsoft created an open source SDK called semantic kernel used for automated AI function chains - github copilot is an example
   - It acts as an alternative to langchain/llamaindex/etc
   - An LLM needs more information than the api is was built on. One way is prompt engineering
   - It is commonly used for creating complex code on the fly, but has a purpose in prompting as well - reading intent of a prompt, and even configuring it in the following ways:

   1) Make the prompt more specific
   2) Add structure to the output with formatting
   3) Provide examples with few-shot prompting (choices)
   4) Tell the AI what to do to avoid doing something wrong
   5) Provide context to the AI (history)
   6) Using message roles in chat completion prompts
   7) Give your AI words of encouragement

c) LLM chaining:
   - The act of connecting an LLM to an external application (a RAG is an example where the external application is a data source)
   - There are various LLM chaining frameworks

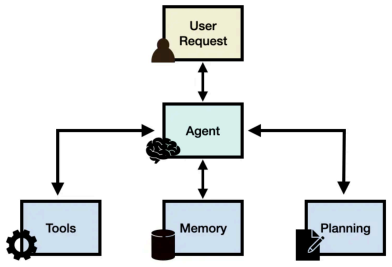
   1) Langchain - good for prototyping due to its breadth and simplicity
   2) LLamaindex - excels at data collection, indexing, and querying, best for semantic search and retrieval
   3) Haystack - best for simpler search and indexing-based LLM applications
   4) AutoGen - best for multi-agent interactions, automation, and conversational prompting

d) LLM agents

a system with complex reasoning capabilities, memory, and the means to execute tasks more complex than a rag.

an llm acts as the 'brain' of the agent.

There are thousands of potential prompts, models, use cases that can be used as agent core available on langchain hubs.



1) User Request - a user question or request
2) Agent/Brain - the agent core acting as coordinator
3) Planning - assists the agent in planning future actions
4) Memory - manages the agent's past behaviors
5) Tools - includes databases, knowledge bases, and external models.

Cloud Computing Services

Amazon EC2

- Offers persistent servers with various instance types tailored to specific requirements.
- Users pay for the uptime of their servers.
- Provides flexibility with instance configurations based on CPU, memory, storage, and GPU.
- Ideal for applications that require continuous availability and resource allocation.

Amazon Lambda

- Eliminates the need to manage servers.
- Charges users only for the compute time consumed by their code.
- Automatically scales to handle incoming requests.
- Best suited for event-driven tasks with sporadic workloads or varying usage patterns.

Additional Functionality
- Llamaindex with PandasQuerying for CSV files as a data source for RAG

Conclusion
- Summary of RAG benefits and implementation strategies
- Comparison of frameworks and tools for specific needs and applications