Natural Language Query Agent

Nanda Kishore Palla

Aditya College of Engineering and Technology

Email: nandakishoreyadavtime@gmail.com

Phone: +91 62813 70959

Github: https://www.github.com/NandaKishoreYadav

Linkedin: https://www.linkedin.com/in/pallanandakishore

Large Language Models (LLM)

A Large Language Model (LLM) refers to a type of artificial intelligence that uses deep learning techniques to process and generate human-like text. These models are trained on vast amounts of text data to understand and generate human language. They can perform a variety of tasks, including language translation, text summarization, question answering, and more. Examples of LLMs include OpenAI's GPT (Generative Pre-trained Transformer) models, which are widely used for natural language understanding and generation tasks.

The current generative AI revolution wouldn't be possible without the so-called large language models (LLMs). Based on transformers, a powerful neural architecture, LLMs are AI systems used to model and process human language. They are called "large" because they have hundreds of millions or even billions of parameters, which are pre-trained using a massive corpus of text data.

Paid LLMs

ChatGPT and Bard, as well as many other popular chatbots, have in common that their underlying LLM are proprietary. That means that they are owned by a company and can only be used by customers after buying a license.

Open Source LLMs

Following rising concerns over the lack of transparency and limited accessibility of proprietary LLMs, mainly controlled by Big Tech, such as Microsoft, Google, and Meta, open-source LLMs promise to make the rapidly growing field of LMMs and generative AI more accessible, transparent, and innovative.

LangChain

LangChain is an open source framework for building applications based on large language models (LLMs). LLMs are large deep-learning models pre-trained on large amounts of data that can generate responses to user queries—for example, answering questions or creating images from text-based prompts. LangChain provides tools and abstractions to improve the customization, accuracy, and relevancy of the information the models generate. For example, developers can use LangChain components to build new prompt chains or customize existing templates. LangChain also includes components that allow LLMs to access new data sets without retraining.

With LangChain, developers can adapt a language model flexibly to specific business contexts by designating steps required to produce the desired outcome.

Chains

Chains are the fundamental principle that holds various AI components in LangChain to provide context-aware responses. A chain is a series of automated actions from the user's query to the model's output. For example, developers can use a chain for:

- Connecting to different data sources.
- Generating unique content.
- Translating multiple languages.
- Answering user queries.

Links

Chains are made of links. Each action that developers string together to form a chained sequence is called a link. With links, developers can divide complex tasks into multiple, smaller tasks. Examples of links include:

- Formatting user input.
- Sending a query to an LLM.
- Retrieving data from cloud storage.
- Translating from one language to another.

In the LangChain framework, a link accepts input from the user and passes it to the LangChain libraries for processing. LangChain also allows link reordering to create different AI workflows.

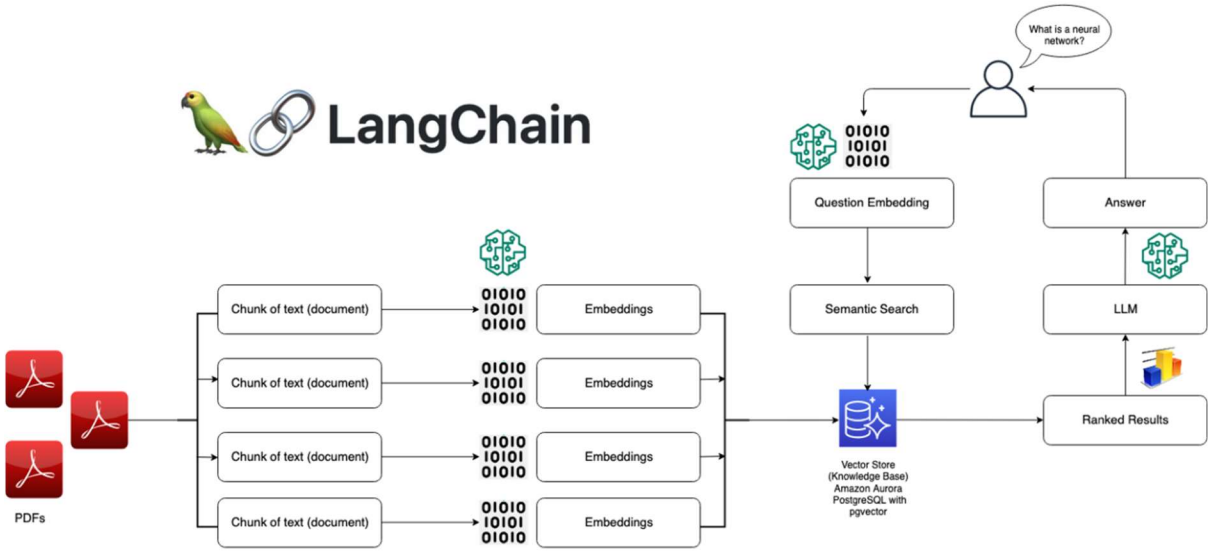

Photo credits : AWS website

Text Splitters

Once you've loaded documents, you'll often want to transform them to better suit your application. The simplest example is you may want to split a long document into smaller chunks that can fit into your model's context window. LangChain has a number of built-in document transformers that make it easy to split, combine, filter, and otherwise manipulate documents.

When you want to deal with long pieces of text, it is necessary to split up that text into chunks. As simple as this sounds, there is a lot of potential complexity here. Ideally, you want to keep the semantically related pieces of text together. What "semantically related" means could depend on the type of text. This notebook showcases several ways to do that.

At a high level, text splitters work as following:

1. Split the text up into small, semantically meaningful chunks (often sentences).
2. Start combining these small chunks into a larger chunk until you reach a certain size (as measured by some function).
3. Once you reach that size, make that chunk its own piece of text and then start creating a new chunk of text with some overlap (to keep context between chunks).

Vector Databases

A vector database indexes and stores vector embeddings for fast retrieval and similarity search, with capabilities like CRUD operations, metadata filtering, horizontal scaling, and serverless.

We're in the midst of the AI revolution. It's upending any industry it touches, promising great innovations - but it also introduces new challenges. Efficient data processing has become more crucial than ever for applications that involve large language models, generative AI, and semantic search.

All of these new applications rely on vector embeddings, a type of vector data representation that carries within it semantic information that's critical for the AI to gain understanding and maintain a long-term memory they can draw upon when executing complex tasks.

Embeddings are generated by AI models (such as Large Language Models) and have many attributes or features, making their representation challenging to manage. In the context of AI and machine learning, these features represent different dimensions of the data that are essential for understanding patterns, relationships, and underlying structures.

That is why we need a specialized database designed specifically for handling this data type. Vector databases like Pinecone fulfill this requirement by offering optimized storage and querying capabilities for embeddings. Vector databases have the capabilities of a traditional database that are absent in standalone vector indexes and the specialization of dealing with vector embeddings, which traditional scalar-based databases lack.

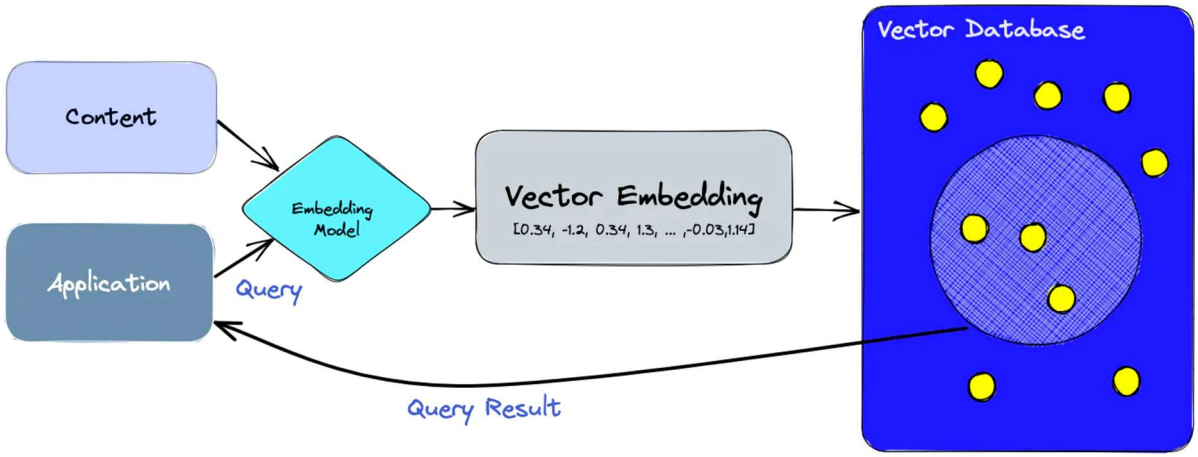

Photo Credits : Pinecone.io

FAISS

FAISS (Facebook AI Similarity Search) is a library that allows developers to quickly search for embeddings of multimedia documents that are similar to each other. It solves limitations of traditional query search engines that are optimized for hash-based searches, and provides more scalable similarity search functions.

Efficient similarity search

With FAISS, developers can search multimedia documents in ways that are inefficient or impossible with standard database engines (SQL). It includes nearest-neighbor search implementations for million-to-billion-scale datasets that optimize the memory-speed-accuracy tradeoff. FAISS aims to offer state-of-the-art performance for all operating points.

FAISS contains algorithms that search in sets of vectors of any size, and also contains supporting code for evaluation and parameter tuning. Some if its most useful algorithms are implemented on the GPU. FAISS is implemented in C++, with an optional Python interface and GPU support via CUDA.

Conversational Memory for LLMs with Langchain

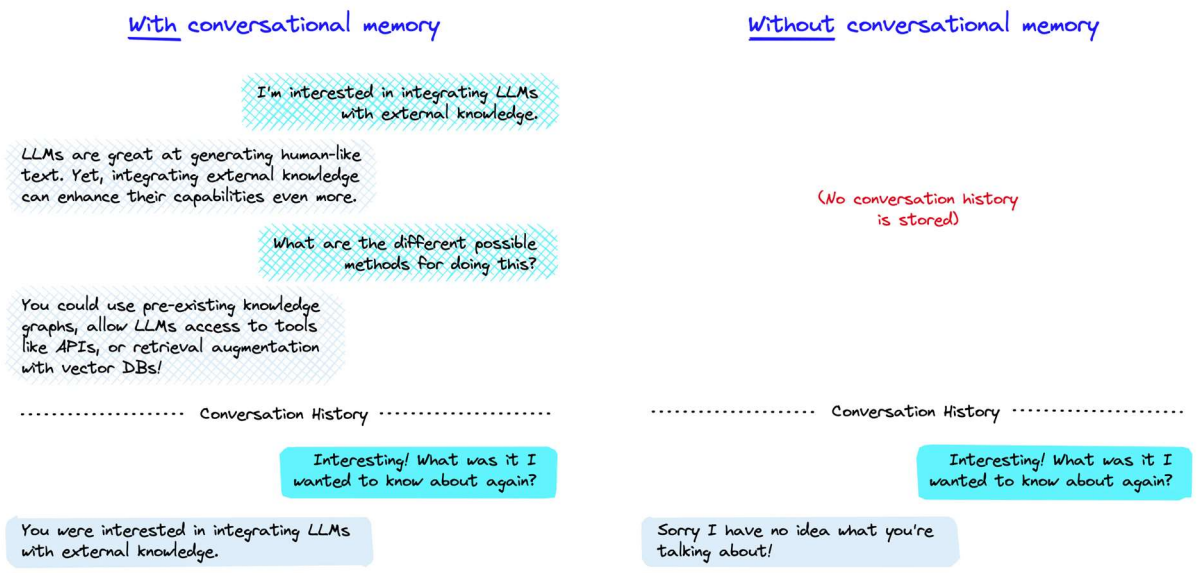

Photo credits: pinecone.io

The memory allows a Large Language Model (LLM) to remember previous interactions with the user. By default, LLMs are stateless — meaning each incoming query is processed independently of other interactions. The only thing that exists for a stateless agent is the current input, nothing else.

There are many applications where remembering previous interactions is very important, such as chatbots. Conversational memory allows us to do that.

There are several ways that we can implement conversational memory. In the context of [LangChain] (/learn/langchain-intro/, they are all built on top of the ConversationChain.

Agents:

Large Language Models (LLMs) trained to perform causal language modeling can tackle a wide range of tasks, but they often struggle with basic tasks like logic, calculation, and search. When prompted in domains in which they do not perform well, they often fail to generate the answer we expect them to.

One approach to overcome this weakness is to create an agent. An agent is a system that uses an LLM as its engine, and it has access to functions called tools. These tools are

functions for performing a task, and they contain all necessary description for the agent to properly use them.

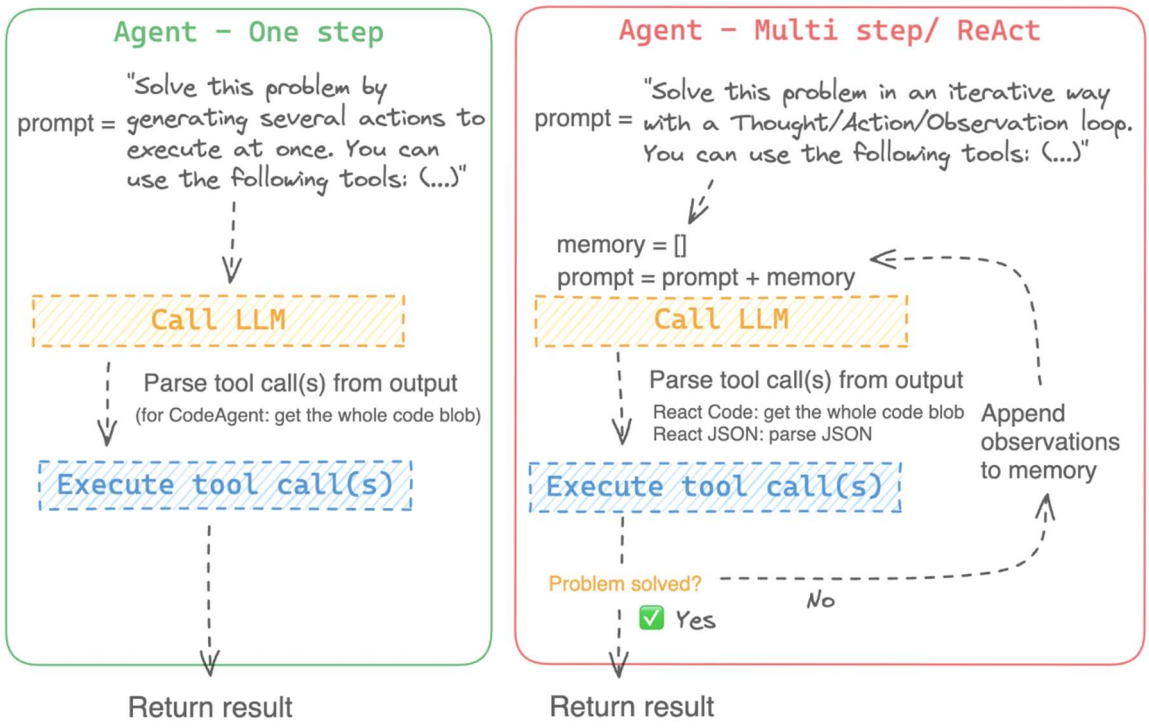

Image credits : huggingface.co

Types of Agents

1. SERP API (Search Engine Results Page API):

- **Definition:** SERP API refers to an Application Programming Interface (API) that allows developers to retrieve search engine results data programmatically.
- **Features:**
  - Provides structured data from search engines like Google, Bing, and others.
  - Supports various search result types, including web pages, images, news, and more.
  - Enables integration into applications for SEO analysis, market research, and content aggregation.

2. **Wikipedia:**

- **Definition:** Wikipedia is a free, collaborative online encyclopedia that allows users to create, edit, and maintain articles on a wide range of topics.
- **Features:**
    - Contains millions of articles in multiple languages, contributed and edited by volunteers worldwide.
    - Offers comprehensive information on diverse subjects, including history, science, culture, and technology.
    - Provides references and citations to ensure content reliability and accuracy.