Langchain

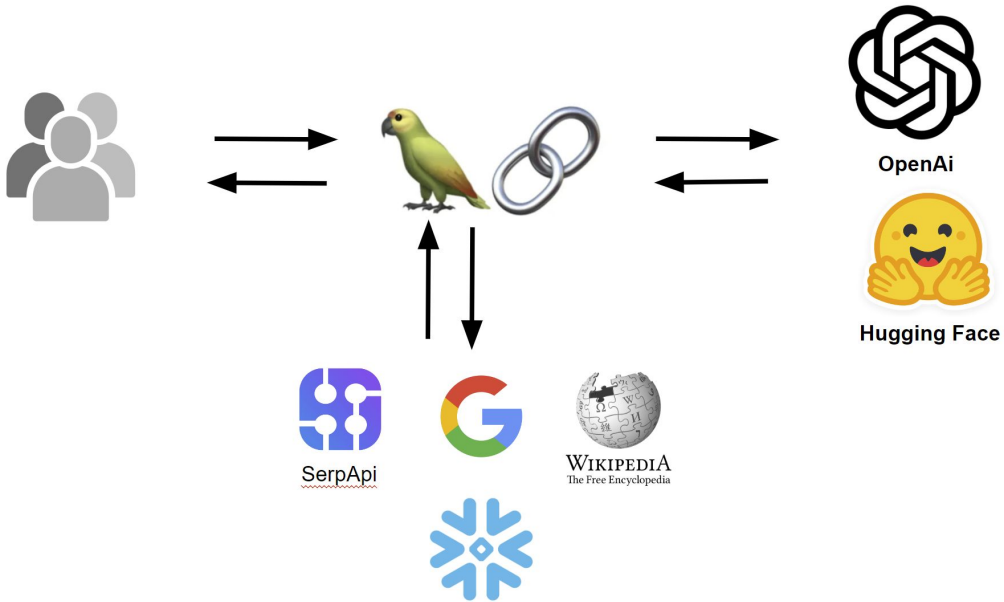
Working with Large Language Models

Agenda

1. Langchain Overview
2. Why Langchain
3. Model I/O
    i. Templates
    ii. Models
    iii. Output Parsers
4. Indexes
    i. Document Loaders
    ii. Vector Stores
5. Chains
6. Agent
7. Memory

Overview

- LangChain is a framework for developing applications powered by language models.
- It simplifies the process of building advanced language model applications.



Overview

- It offers a suite of tools, components, and interfaces for managing interactions with language models.
  - It makes easy to chain together multiple LLMs and Prompts to create complex applications.
- It integrates additional resources like APIs and databases.
- It is an extensible framework that can be customized to meet the specific needs of your application.
- It is a scalable framework that can be used to create applications that can handle large volumes of data and traffic.

Why Langchain?

- Large Language Model have token limitation.
- Large Language Model trained on historical data, and doesn’t have latest information. LangChain addresses this limitation by allowing the LLM to access external data
- LLMs trained on large generalized data that lack context to task in hand. LangChain addresses by providing contextual information.
- Include external data sources.

Prompt Template

- A prompt template refers to a reproducible way to generate a prompt. It contains a text string ("the template"), that can take in a set of parameters from the end user and generates a prompt.
- A prompt template can contain:
  - Instructions to the language model
  - A set of few shot examples to help the language model generate a better response
  - A question to the language model
- A prompt template helps to improve the accuracy and consistency of the output from LLMs.

Language models

LangChain provides interfaces and integrations for two types of models:

- LLMs: Models that take a text string as input and return a text string.
- Chat models: Models that are backed by a language model but take a list of Chat Messages as input and return a Chat Message.

Output Parsers

- Output parsers are classes that help structure language model responses, rather than just text.
- There are 2 main methods,
  - "Get format instructions": A method which returns a string containing instructions for how the output of a language model should be formatted.
  - "Parse": A method which takes in a string and parses it into some structure.
- Output parsers supported by Langchain : List, Datetime , enum, json

Indexes

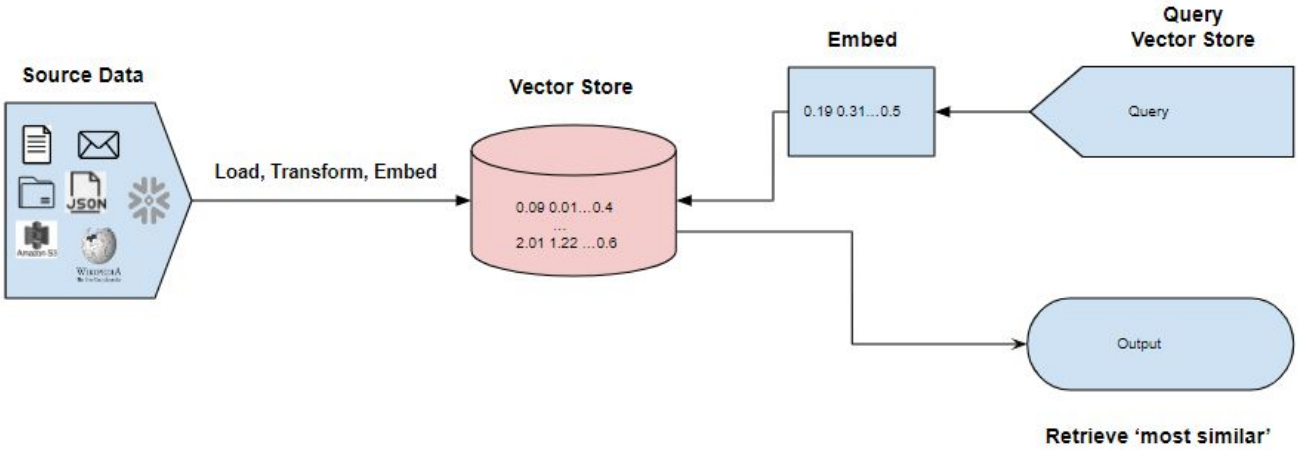
- Indexes refer to ways to structure documents so that LLMs can best interact with them.
  - **Document Loaders:** Classes responsible for loading documents from various sources.
  - **Text Splitters:** Classes responsible for splitting text into smaller chunks.
  - **VectorStores:** The most common type of index. One that relies on embeddings.
  - **Retrievers:** Interface for fetching relevant documents to combine with language models.

Document Loaders

- Use document loaders to load data from a source.
- Langchain has different Integrations, which allows you to connect with numerous source types, Wikipedia, Twitter, youtube, Snowflake, Clouds - AWS/GCP/Azure.

Vector Stores

- It creates numerical embeddings for each document, and store the resulting embedding vectors in Vector Store
- At query time to embed the unstructured query and retrieve the embedding vectors that are 'most similar' to the embedded query.
- There are multiple integrations option provided by Langhchain. Ex. PineCone, Chroma, Redis, FAISS, Elastic Search etc.



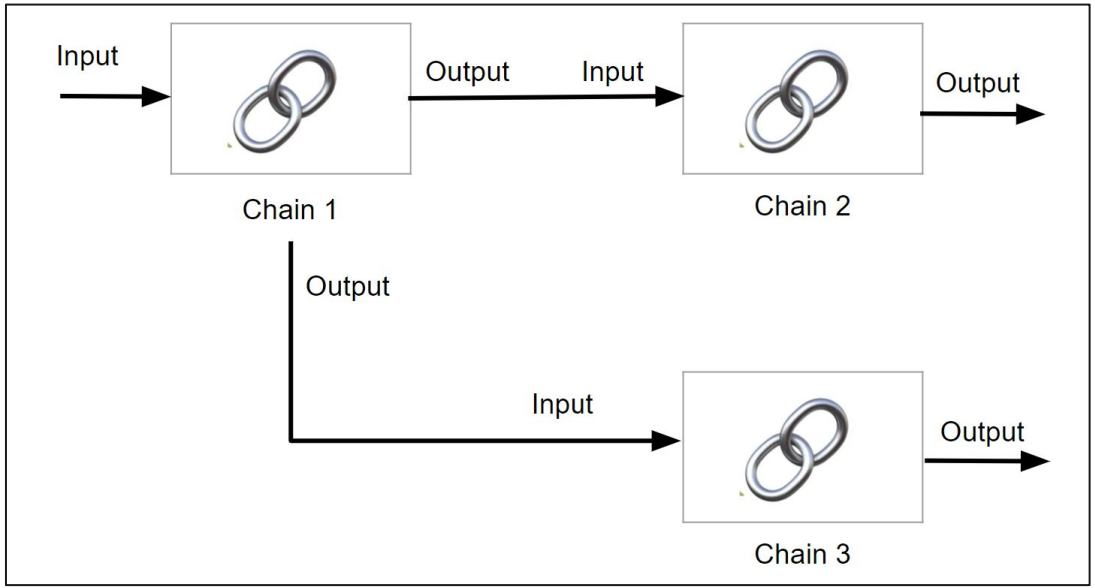
Chains

- Chains allow us to combine multiple components together to create a single, coherent application.

  Ex. Bot to find currency of different country, and compare it with Indian 100Rs.

  > What is the currency of USA? How much dollars I need for 100 INR?.

Chains - types



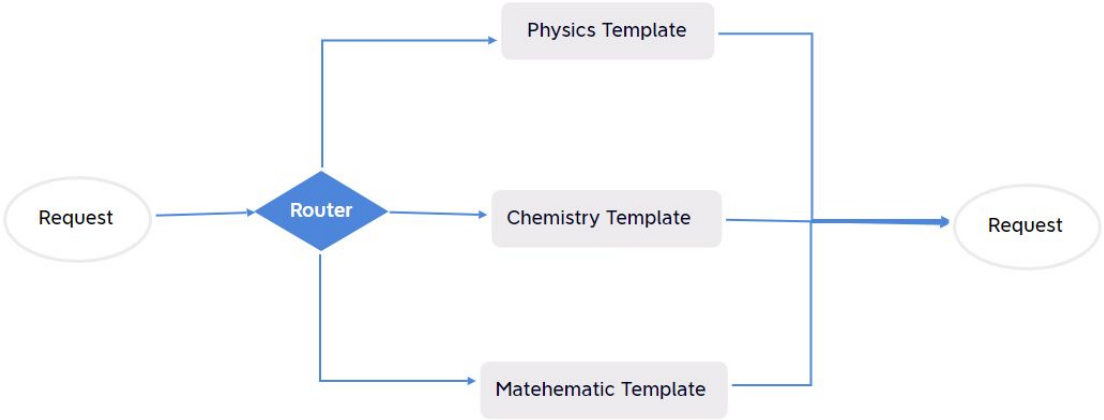
Sequential Chain

A sequential chain works by combining two or more chains. For example you can take the output from one chain and pass it across to next chains sequentially.

Chains - types

Router Chain

Router Chain is useful when you have multiple chains for different tasks and you wish to invoke them based on the input provided.



Chains - types

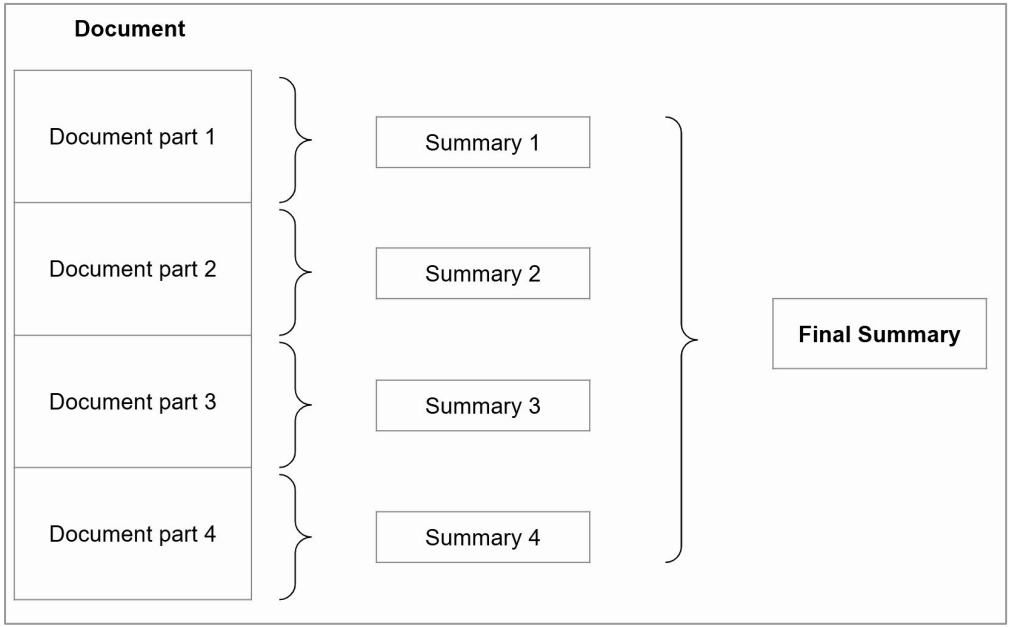
LoadSummarize chain

This chain helps to summarization over big documents.

It first splits the entire input into small chunks using a text splitter.

Create a summary of each of these chunks.

Once we get a summary of each, the it creates a summary over these summaries and provides an output



Agent

- Some applications require a flexible chain of calls to LLMs and other tools based on user input.
- An agent has access to a suite of tools, and determines which ones to use depending on the user input. Agents can use multiple tools, and use the output of one tool as the input to the next.
- There are two main types of agents:
  - Action agents: at each timestep, decide on the next action using the outputs of all previous actions
  - Plan-and-execute agents: decide on the full sequence of actions up front, then execute them all without updating the plan

Agent - types

- **Zero-shot ReAct:** This agent uses the ReAct framework to determine which tool to use based solely on the tool's description. This agent requires that a description is provided for each tool.
- **OpenAI Functions:** Certain OpenAI models (like gpt-3.5-turbo-0613 and gpt-4-0613) have been explicitly fine-tuned to detect when a function should to be called and respond with the inputs that should be passed to the function.
- **Conversational:**This agent is designed to be used in conversational settings. The prompt is designed to make the agent helpful and conversational.
- **Self ask with search:** This agent utilizes a single tool that should be named Intermediate Answer. This tool should be able to lookup factual answers to questions.

Memory

- The chains and agents are stateless, but for many applications, it's necessary to reference past interactions.

  Ex. chatbots.

- The Memory module enables to maintain the application state.
- The base Memory interface is simple: it lets you update the state given the latest run inputs and outputs and it enables you to modify (or contextualize) the following input using the stored state.
  - The simplest of these is a buffer memory which just prepends the last few inputs/outputs to the current input.

UseCases

- **Chatbots:** use it to create chatbots that can understand and respond to user input in a natural way.
- **Question Answering Over Documents:** use it to create question answering systems that can answer questions in a comprehensive and informative way.
- **Data analysis:** use it to analyze data by generating text summaries, creating visualizations, or answering questions about the data.
- **Summarization:** Use it to summarize documents without worrying about
- **Personal Assistants**

References

- https://python.langchain.com/docs/get_started/introduction.html
- https://github.com/hwchase17/langchain
- https://www.youtube.com/watch?v=_v_fgW2SkkQ&list=PLqZXAkvF1bPNQER9mLmDbntNfSpzdDIU5
- https://www.youtube.com/watch?v=aywZrzNaKjs
- https://learn.deeplearning.ai/langchain/lesson/1/introduction