Building Generative AI Applications with LangChain and OpenAI API

Avikumar Talaviya
27 Jul, 2023 • 8 min read

Introduction

Generative AI is leading the latest tech wave in the industry. Generative AI Applications like image generation, text generation, summarization, and question-and-answer bots, to name a few, are booming. As OpenAI recently led the large language model wave, many startups developed tools and frameworks to allow developers to build innovative applications using these LLMs. One such tool is LangChain, a framework to develop applications powered by LLMs with composability and reliability. LangChain has become the go-to tool for AI developers worldwide to build generative AI applications. LangChain also allows for connecting external data sources and integration with many LLMs available on the market. Apart from this, LLM-powered apps require a vector storage database to store the data they will retrieve later on. In this blog, we will learn about LangChain and Its functions by building an application pipeline with OpenAI API and ChromaDB.

- Learn the fundamentals of LangChain to build a generative AI pipeline
- Text embedding using Open source models and vector storage databases like Chromadb
- Learn to use OpenAI APIs with LangChain to integrate LLMs into your application

This article was published as a part of the Data Science Blogathon.

Table of contents

Overview of a LangChain

LangChain has recently become a popular framework for large language model applications. LangChain provides a sophisticated framework to interact with LLMs, external data sources, prompts, and User Interfaces.

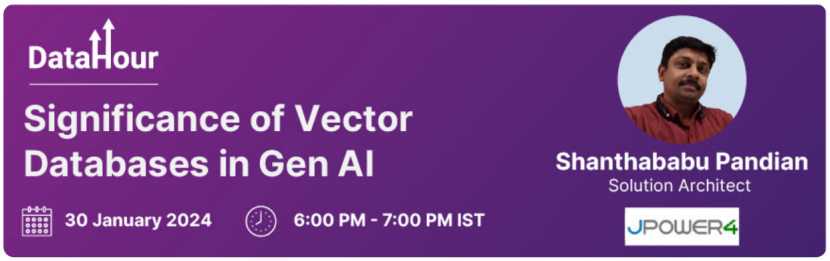
Value Propositions of LangChain

The main value propositions of the LangChain are:

- **Components:** These are the abstractions needed to work with language models. Components are modular and easy to use for many LLM use cases.
- **Off-the-shelf chains:** A structured assembly of various components and modules to accomplish a specific task, such as summarization, Q&A, etc.

Project Details

LangChain is an Open source project, and since its launch, the project has garnered over 54K+ Github stars, which shows the popularity and acceptability of the project.

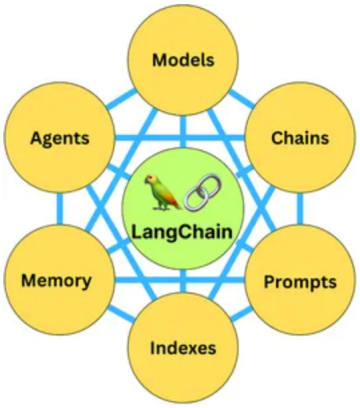


The project readme file describes the framework in something like this:

Large language models (LLMs) are emerging as a transformative technology, enabling developers to build applications they previously could not. However, using these LLMs in isolation is often insufficient for creating a truly powerful app-the real power comes when you combine them with other sources of computation or knowledge.

Source: Project Repo

Clearly, It defines the framework's purpose with its aims to assist in developing such applications where user knowledge is leveraged.



LangChain Components (source: ByteByteGo)

LangChain has six main components to build LLM applications: model I/O, Data connections, Chains, Memory, Agents, and Callbacks. The framework also allows integration with many tools to develop full-stack applications, such as OpenAI, Huggingface Transformers, and Vectors stores like Pinecone and chromadb, among others.

Comprehensive Explanation of Components:

1. **Model I/O:** Interface with language models. It consists of Prompts, Models, and Output parsers
2. **Data connection:** Interface with application-specific data sources with data transformers, text splitters, vector stores, and retrievers
3. **Chains:** Construct a sequence of calls with other components of the AI application. some examples of chains are sequential chains, summarization chain, and Retrieval Q&A Chains
4. **Agents:** LangChain provides Agents which allow applications to utilize a dynamic chain of calls to various tools, including LLMs, based on user input.
5. **Memory:** Persist application state between runs of a chain
6. **Callbacks:** Log and stream steps of sequential chains in order to run the chains efficiently and monitor the resources consumption

Let's now look at some of the use cases for LangChain.

1. **Question answering or chat over specific documents**
2. **Chatbots**
3. **Summarization**
4. **Agents**

5. **Interacting with APIs**

These are a few of the many use cases. We will learn and develop a semantic search application for question answering on specific documents using OpenAI APIs and ChromaDB, an open-source vector database. To learn more about the LangChain framework, I highly recommend reading the official documentation. (Link: here)

Environment Set Up and Loading Documents

Now, we will set up an environment for our semantic search application using OpenAI's LLM APIs to answer users' questions over a set of documents. We are using sample documents in this article, but you can use your document to build a question-answering application. First, we need to install the following libraries:

Installing the Project Dependencies

```
# install openai, langchain, sentense transfoers and other dependenci
!pip install openai langchain sentence_transformers -q
!pip install unstructured -q

# install the environment dependencies
!pip install pydantic==1.10.8
!pip install typing-inspect==0.8.0 typing_extensions==4.5.
!pip install chromadb==0.3.26
```

LangChain requires some environment dependencies with a specific version, such as pydantic, typing extensions, and ChromaDB. Once the installation is complete, you can run the following code in your colab or any other notebook environment.

LangChain Document Loader

LangChain provides document loader classes to load documents from user input or database. It supports various file formats, such as HTML, JSON, CSV, etc. We have a few text files that we will use in our use case. You can find the files in the GitHub repo. (GitHub repo—Link)

```
# import langchain dir loader from document loaders
from langchain.document_loaders import DirectoryLoader

# directory path
directory = '/content/pets'

# function to load the text docs
```

```
def load_docs(directory):
  loader = DirectoryLoader(directory)
  documents = loader.load()
  return documents

documents = load_docs(directory)
len(documents)

--------------------------[Output]---------------------------------
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
5
```

Once the data is loaded, we will use a text splitter to split the text documents into the fixed size of chunks to store them in the vector database. LangChain offers multiple text splitters such as split by character, split by code, etc.

```
# use text splitter to split text in chunks
from langchain.text_splitter import RecursiveCharacterTextSplitter

# split the docs into chunks using recursive character splitter
def split_docs(documents,chunk_size=1000,chunk_overlap=20):
  text_splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_siz
  docs = text_splitter.split_documents(documents)
  return docs

# store the splitte documnets in docs variable
docs = split_docs(documents)
```

Once the documents are converted into chunks, we will embed them into a vector using open-source embedding models in the next section.

Text Embedding Using LangChain and Open Source Model

Embedding texts is the most important concept in the LLM application development pipeline. All the text documents need to be vectorized before they can be processed for tasks like semantic search, summarization, etc. We will use the open-source sentence-transformer model "***all-MiniLM-L6-v2***" for text embeddings. Once the documents are embedded, we can store them in the open-source vector database ChromaDB to perform a semantic search. Let's look at the hands-on code example.

```
# embeddings using langchain
from langchain.embeddings import SentenceTransformerEmbeddings
embeddings = SentenceTransformerEmbeddings(model_name="all-MiniLM-L6-

# using chromadb as a vector store and storing the docs in it
```

```
from langchain.vectorstores import Chroma
db = Chroma.from_documents(docs, embeddings)

# Doing similarity search  using query
query = "What are the different kinds of pets people commonly own?"
matching_docs = db.similarity_search(query)

matching_docs[0]

--------------------------[output]-----------------------------------
Document(page_content='Pet animals come in all shapes and sizes,
each suited to different lifestyles and home environments.
Dogs and cats are the most common, known for their companionship
and unique personalities. Small mammals like hamsters, guinea pigs,
and rabbits are often chosen for their low maintenance needs.
 Birds offer beauty and song, and reptiles like turtles and
lizards can make intriguing pets. Even fish, with their calming prese
can be wonderful pets.',
metadata={'source': '/content/pets/Different Types of Pet Animals.txt
```

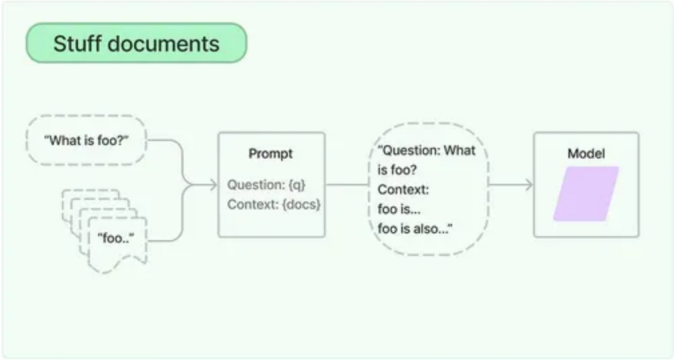
In the above code, we use embeddings to store in *ChromaDB*, which supports in-memory storage. so we can query the database to get the answer from our text documents. We asked to know about different kinds of pets commonly owned by people, and it resulted in a correct answer with the source of the answer.

Generative AI applications Using OpenAI APIs, ChromaDB, and LangChain

Semantic search Q&A Using LangChain and OpenAI APIs

This pipeline entails interpreting the intent and context of search terms and documents to produce more precise search results. It can increase search accuracy by comprehending user intent, examining the connections between words and concepts, and producing context-aware search results by utilizing attention mechanisms in natural language processing (NLP).

LangChain offers an OpenAI chat interface to call the model APIs into your application and create a question/answer pipeline that answers users' queries based on given context or input documents. It basically performs a vectorized search to find the most similar answer to the question. (Refer to the below flow chart.)



Context-based search pipeline (source: LangChain Docs)

```
# insert an openai key below parameter
import os
os.environ["OPENAI_API_KEY"] = "YOUR-OPENAI-KEY"

# load the LLM model
from langchain.chat_models import ChatOpenAI
model_name = "gpt-3.5-turbo"
llm = ChatOpenAI(model_name=model_name)

# Using q&a chain to get the answer for our query
from langchain.chains.question_answering import load_qa_chain
chain = load_qa_chain(llm, chain_type="stuff",verbose=True)

# write your query and perform similarity search to generate an answe
query = "What are the emotional benefits of owning a pet?"
matching_docs = db.similarity_search(query)
answer =  chain.run(input_documents=matching_docs, question=query)
answer

-----------------------------------[Results]-------------------------
'Owning a pet can provide numerous emotional benefits. Pets offer
 companionship and can help reduce feelings of loneliness and isolati
They provide unconditional love and support, which can boost mood and
 well-being. Interacting with pets, such as petting or playing with t
has been shown to decrease levels of stress hormones and increase the
 release of oxytocin, a hormone associated with bonding and relaxatio
Pets also offer a sense of purpose and responsibility, as taking care
them can give a sense of fulfillment and provide a distraction from d
stressors. Additionally, the bond between pets and their owners can p
 a sense of stability and consistency during times of personal or soc
stress.'
```

The above code, calls the "***gpt-3.5-turbo***" model API using LangChain's ChatOpenAI() function and creates a q&a chain for answering our query. for more detailed information on code, you can visit LangChain's official documentation (here) and Github code notebook (here)

Generate Company Names Using LLMChain

The other use-case of the large language model is to generate company names using **LLMChain, OpenAI LLM, and PromptTemplate** by LangChain. you can generate company or product names based on the given description as a prompt. Refer to the below code:

```
# import necessary components from langchain
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain

# LLM by OpenAI
llm = OpenAI(temperature=0.9)

# Write a prompt using PromptTemplate
prompt = PromptTemplate(
    input_variables=["product"],
    template="What is a good name for a company that makes {product}?
)

# create chain using LLMChain
chain = LLMChain(llm=llm, prompt=prompt)

# Run the chain only specifying the input variable.
print(chain.run("colorful socks"))

Output>>>    Colorful Toes Co.
```

Text Documents Summarization Using LangChain Chains

Chains also allow the development of text summarization applications which can be helpful in the Legal industry to summarize vast amounts of legal documents to speed up the judicial process. Refer to the example code below using LangChain's function:

```
# from langChain import load_suumarize_chain function and OpenAI llm
from langchain.llms import OpenAI
from langchain.chains.summarize import load_summarize_chain

# LLM by OpenAI
llm = OpenAI(temperature=0.9)

# create a chain instance with OpenAI LLM with map_reduce chain type
chain = load_summarize_chain(llm, chain_type="map_reduce")
chain.run(docs)
```

LangChain's easy-to-use interface unlocks many different applications and solves many problems for end-users. As you can see, with simple few lines of code, we can leverage the power of LLMs to summarize any data from any sources on the web.

Conclusion

In conclusion, the blog post explored the exciting domain of building generative AI applications using LangChain and OpenAI APIs. We saw an overview of LangChain, its various components, and use cases for LLM applications. Generative AI has revolutionized various domains, allowing us to generate realistic text, images, videos, and more.

Semantic search is one application used to build question-and-answer applications using OpenAI LLMs like GPT-3.5 and GPT-4. Let's look at the key takeaways from this blog:

1. We learned about a brief overview of LangChain—An open source framework to build LLM-powered applications.
2. We learned to use LangChain and ChromaDB—A vector database to store embeddings for similarity search applications.
3. Finally, we learned about OpenAI LLM APIs to build generative AI applications using LangChain.

Frequently Asked Questions

Q1. What is a generative AI?

A. Generative AI is a machine-learning technology that learns from much-unstructured data to generate new text, images, music, and even videos.

Q2. What is a semantic search?

A. Semantic search applications entail interpreting the intent and context of search terms and documents to produce precise search results given a certain query by users.

Q3. What is a LangChain, and what are its key features?

A. LangChain is a popular framework in large language model application development. LangChain provides a framework to interact with LLMs, external data sources, prompts, and User Interfaces.

Q4. What are the benefits of LangChain?

A. LangChain enables connection with a wide range of external data sources. It also integrates LLM in various use cases, such as a chatbot, summarization, and code generation.

Q5. What databases are supported by LangChain?

A. LangChain supports vector databases like ChromaDB and Pinecone and structured databases like MS SQL, MySQL, MariaDB, PostgreSQL, Oracle SQL, etc.

Related

Revamp Data Analysis: OpenAI, LangChain & LlamaIndex for Easy Extraction

Mastering Prompt Engineering for LLM Applications with LangChain

Fundamental Principles of Langchain in LLM Based Application Development

AI API Applications blogathon documents framework Generative AI LLM OpenAI vector

Avikumar Talaviya
27 Jul 2023

Beginner Database Generative AI Github Large Language Models

Related Articles

Revamp Data Analysis: OpenAI, LangChain & LlamaIndex for Easy ...

Using OpenAI's API combined with LangChain and LlamaIndex to extract valuable in...

API blogathon

Deepak K
20 Jun, 2023 • 6 min read

Mastering Prompt Engineering for LLM Applications with LangChain

Equip readers with the knowledge and tools to craft dynamic and context-aware pr...

API Applications

Sai Nitish ...
28 Jul, 2023 • 8 min read

Fundamental Principles of Langchain in LLM Based Application Develo...

Learn all about LangChain - what it is, what are its components, and how to use ...

AI API

Saptarshi D...
30 Sep, 2023 • 8 min read

Scribe, Shine, Succeed →

Write, captivate, and earn accolades and rewards for your work

- Reach a Global Audience
- Get Expert Feedback
- Build Your Brand & Audience
- Cash In on Your Knowledge
- Join a Thriving Community
- Level Up Your Data Science Game

Arnab Mondal
15

Prateek Majumder
68