



LangChain

Developing LLM Apps with LangChain

LLMs like GPT are great at answering questions about data they've been trained on...but what if you want to ask it questions about data it hasn't been trained on? For example, maybe you want to ask them about information from after their training cut-off date, or information from non-public documents?

One of the best ways to do this is inputting the information, even large amounts of information such books and documents, into the model. And that's exactly what this course will teach you from scratch!

In this course you'll learn how to build state-of-the-art LLM-powered applications with LangChain, Pinecone, OpenAI, and Python! We'll build together, step-by-step, line-by-line. This will be a learning-by-doing experience.

Tools to be used

Python, OpenAI, LangChain, Vector Embeddings, Pinecone, Chroma.

Resources

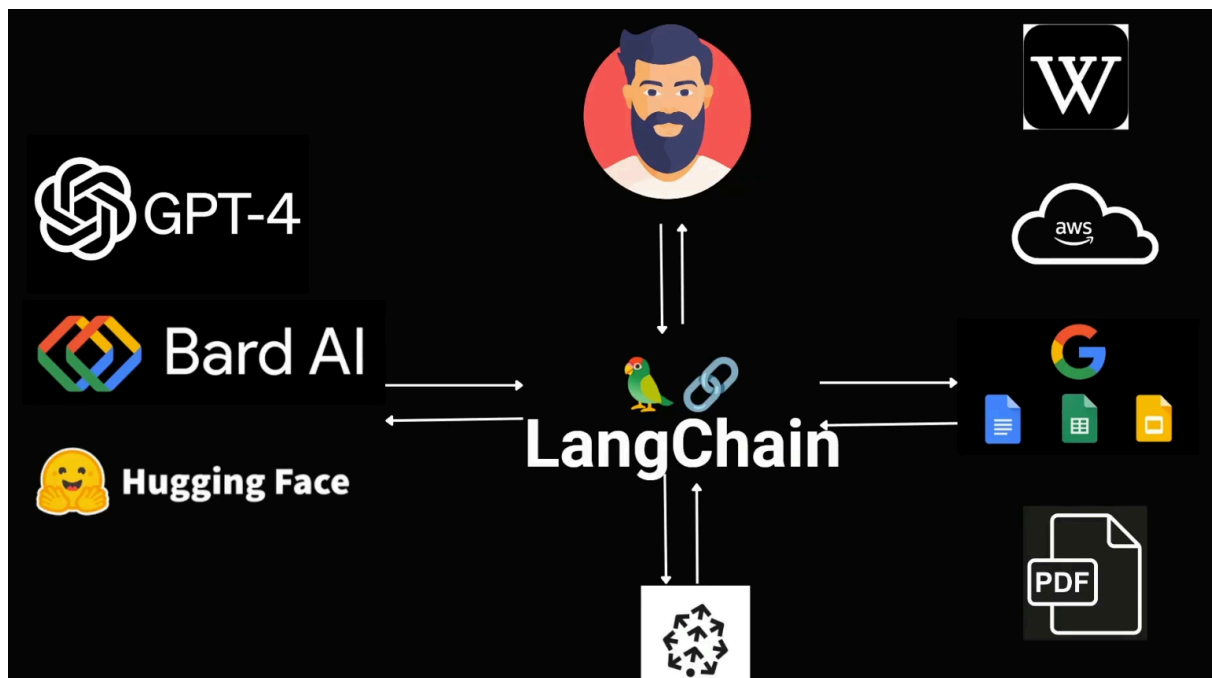
<https://drive.google.com/drive/folders/1OE8Q-QQdzKdVRbkVqItOwX5v0ckcIGTI>

What's LangChain?

LangChain is an open-source framework that allows developers working with AI to combine large language models (LLMs) like GPT-4 with external sources of computation and data.

It makes it easy to build and deploy AI applications that are both scalable and performant. LangChain is a great entry point into the AI field for individuals from diverse backgrounds and enables the deployment of AI as a service. It has a virtually infinite number of practical use cases.

LLMs have an impressive general knowledge but are limited to their training data. LangChain allows you to connect an LLM to your own sources of data. LangChain is data-aware and agentic-aware.



LangChain Use-Cases

- Chat Bots
- Question Answering Systems
- Summarization Tools

LangChain Main Concepts

- LangChain Components

- **LLM Wrappers:** Enable connections to large language models, including those from the Hugging Face Hub.
- **Prompt Templates:** Create dynamic prompts that serve as input for LLMs.
- **Indexes:** Extract relevant information for LLMs to process.
- **Memory:** Handle data storage and retrieval during conversations.
 - **Short-term** → Manages data flow within a single conversation.
 - **Long-term** → Handles information transfer between different conversations.
- **Chains:** Combine multiple components to create complete LLM applications for specific tasks.
- **Agents:** Connect LLMs with external APIs, guide decision-making, and manage action cycles—taking actions, observing results, and repeating until the task is complete.

Caching in LangChain

Caching is the practice of storing frequently accessed data or results in a temporary, faster storage layer.

Caching optimizes interactions with LLMs by reducing API calls and speeding up applications, resulting in a more efficient user experience.

LangChain provides an optional caching layer for LLMs and there are two caching options: **InMemoryCache** and **SQLiteCache**.

- **InMemoryCache:** stores results in RAM during the current session; it's fast but data is lost when the program ends.
- **SQLiteCache:** saves responses locally in a .sqlite file; results persist between runs of the application.

Streaming in LangChain

Streaming refers to the process of delivering the response in a continuous stream of data instead of sending the entire response at once.

This allows the user to receive the response piece by piece as it is generated, which can improve the user experience and reduce the overall latency.

Prompt Templates

A prompt refers to the input to the model. Prompt templates are a way to create dynamic prompts for LLMs.

A prompt template takes a piece of text and injects a user's input into that piece of text. In LangChain there are **PromptTemplates** and **ChatPromptTemplates**.

- **PromptTemplate**: generates a single formatted string for completion models (e.g., `text-davinci-003`). Are used for tasks that involve generating text such as answering or completing sentences.
- **ChatPromptTemplate**: builds structured chat messages (e.g., system, user, assistant) for chat-based models like GPT-4. Are specifically designed for tasks that involve engaging in conversations.

LangChain Chains

Chains are a series of steps and actions. Allow us to combine multiple components together to solve a specific task and build an entire LLM application. There are three types: simple, sequential and custom chains.

With sequential chains, you can make a series of calls to one or more LLMs. You can take the output from one chain and use it as the input to another chain. There are two types of sequential chains:

- **SimpleSequentialChain**: represents a series of chains, where each individual chain has a single input and a single output, and the output of one step is used as input to the next.
- General form of sequential chains.

LangChain Agents

LangChain Agents are components that use an LLM to decide which tools or actions to take based on user input, enabling dynamic and multi-step task execution.

Agent is a class that uses an LLM to choose a sequence of actions to take. In Chains, a sequence of actions is hardcoded. In Agents, a language model is used as a reasoning engine to determine which actions to take and in which order.

Agents select and use **Tools** and **Toolkits** for actions.

- **Agents** reason and decide.
- **Tools** execute actions.
- **Toolkits** group tools by context or purpose.

An example ...

Let's say we want an agent that answers natural language questions by querying a SQL database.

In this example:

- The Agent interprets the question and plans the steps.
- The Tools are specific SQL query functions.
- The Toolkit provides those tools in a ready-to-use package.

```
from langchain.agents import initialize_agent,
from langchain.agents.agent_toolkits import S
from langchain.tools import Tool
from langchain.utilities import SQLDatabase
from langchain_openai import OpenAI
```

```
# 1. Initialize the database connection
db = SQLDatabase.from_uri("sqlite:///my_data
```

```
# 2. Create the toolkit with tools to interact wit
toolkit = SQLDatabaseToolkit(db=db, llm=Ope
```

```
# 3. Extract tools from the toolkit
tools = toolkit.get_tools()
```

```
# 4. Initialize the agent with those tools
agent = initialize_agent(
    tools=tools,
    llm=OpenAI(temperature=0),
    agent=AgentType.ZERO_SHOT_REACT_DES
    verbose=True
)
```

```
# 5. Ask a natural language question — the ag
response = agent.run("What are the top 5 pro
print(response)
```

LangChain Tools

Are like specialized apps for your LLM. They are tiny code modules that allow it to access information and services. These tools connect your LLM to search

engines, databases, APIs, and more, expanding its knowledge and capabilities.

ReAct - Reasoning and Acting

ReAct is a new approach that combines reasoning (chain-of-thoughts prompting) and acting capabilities of LLMs.


With ReAct, LLMs generate **reasoning traces** and **task-specific actions** in an interleaved manner.

A model can reason about a task and then take actions in the real world to gather more information and complete the task. This combination of reasoning and acting allows ReAct to overcome some of the limitations of existing LLMs, such as hallucinations and error propagations.

LangChain Agent: Tools + Chains

HX-AQuintero/LangChain-Deep-Dive

Code and notes from the Developing LLM Apps with
LangChain ZTM Course - HX-AQuintero/LangChain-Deep-Dive

 <https://github.com/HX-AQuintero/LangChain-Deep-Dive>

HX-AQuintero/**LangChain-Deep-Dive**

Code and notes from the Developing LLM Apps with
LangChain ZTM Course



 0  0  0  0
Contributors Issues Stars Forks

