**LangChain Crash Course: Your Go-To Guide for Building LLM Apps! 💡  
Reference video:**

[**https://www.youtube.com/watch?v=nAmC7SoVLd8&list=PLeo1K3hjS3uu0N\_0W6giDXzZIcB07Ng\_F**](https://www.youtube.com/watch?v=nAmC7SoVLd8&list=PLeo1K3hjS3uu0N_0W6giDXzZIcB07Ng_F)

1. **Welcome to LangChain!**
   * **What's the Big Deal?**
   * **Why We Need LangChain (The Problems it Solves)**
2. **Getting Your Hands Dirty: Setup Time!**
   * **OpenAI Account & API Key (Your Secret Sauce 🤫)**
   * **Installation Station (Pip It Up! 🐍)**
3. **LangChain's Core Superpowers (Concepts Explained!)**
   * **LLMs: The Brains of the Operation 🧠**
   * **Prompt Templates: Smart Fill-in-the-Blanks 📝**
   * **Chains: Connecting the Dots 🔗**
     + **LLMChain: The Basic Link**
     + **Simple Sequential Chain: One Step After Another**
     + **Sequential Chain: The Multi-Tasker**
4. **Building Your First App: Restaurant Idea Generator 🍽️**
   * **Using Streamlit for a Quick UI (No React Needed!)**
   * **Putting it All Together (Code Snippets & Logic)**
5. **Agents: The Smart Decision Makers 🤖**
   * **What Agents Do**
   * **Tools: Giving Agents Abilities (Wikipedia, Math, Google Search)**
6. **Memory: So Your AI Doesn't Forget (Like Us Sometimes! 😉)**
   * **Why Memory Matters**
   * **ConversationBufferMemory: The Full Story**
   * **ConversationBufferWindowMemory: Just the Highlights**
7. **Wrapping It Up!**

**1. Welcome to LangChain! 👋**

**What's the Big Deal?**

* Imagine building awesome apps that use super-smart AI models (Large Language Models or LLMs like ChatGPT's brain, GPT-3.5/4). **LangChain** is like your friendly toolbox 🛠️ that makes this process way easier and faster! It's a framework that lets you put together all the pieces needed to build powerful LLM applications.

**Why We Need LangChain (The Problems it Solves)**

* You might think, "Why not just use the OpenAI API directly?" Good question! But there are a few snags:
  + **💰 Cost, Cost, Cost!** Calling the OpenAI API isn't free. Every "token" (think of it as a word or part of a word) costs money. For a startup or a student on a budget, this can add up fast.
  + **🗓️ Old News (Sometimes)!** ChatGPT's knowledge has a cut-off date (like September 2021 for the video's recording). It can't tell you about the latest happenings.
  + **🔒 No Secret Sauce Access!** These LLMs don't know anything about your personal data or your company's internal information. So, asking it about your last month's employee count won't get you an answer.
* **Enter LangChain, the Problem Solver!** It helps by:
  + Letting you swap out expensive OpenAI models for cheaper, open-source ones if you want to save cash.
  + Connecting to real-time information sources like Google Search or Wikipedia to get up-to-date data.
  + Even linking up with your own databases to use your private information.

**2. Getting Your Hands Dirty: Setup Time! 🛠️**

**OpenAI Account & API Key (Your Secret Sauce 🤫)**

* To play with LangChain and OpenAI, you'll need an **API key** from OpenAI.
* **How to get it:**
  1. Go to the OpenAI website.
  2. Log in (or create an account using Google or email).
  3. Once logged in, head to your **API keys** section (usually under "Manage Account").
  4. You'll see a key that looks like SK-hyphen-something. This is your password for using the API, so keep it super secret!
  5. You can generate new keys for different projects. **Important:** Copy it somewhere safe immediately after creation, as you won't be able to see it again.
* **Using the Key in Your Code:**
  1. You can set it as an **environment variable** in your Python script:

Python

import os

os.environ["OPENAI\_API\_KEY"] = "YOUR\_API\_KEY\_HERE" # Replace with your actual key

* 1. The video creator uses a separate secret\_key.py file to store their key for security, then imports it.

**Installation Station (Pip It Up! 🐍)**

* Before writing any code, you need to install the necessary libraries. Open your terminal or command prompt and run these commands:

Bash

pip install langchain

pip install openai

**3. LangChain's Core Superpowers (Concepts Explained!) 💡**

**LLMs: The Brains of the Operation 🧠**

* In LangChain, you interact with LLMs. While you can choose various models, the video focuses on **OpenAI's LLMs** because they're top-notch.
* **Code Example (Importing and Initializing OpenAI LLM):**

Python

from langchain.llms import OpenAI

import os # Make sure your API key is set as an environment variable or imported

# Initialize the LLM

# The 'temperature' parameter controls creativity:

# 0 = very safe, predictable (not creative)

# 1 = very risky, creative (might make mistakes)

# 0.6 or 0.7 is a good balance for creativity

llm = OpenAI(temperature=0.7)

# Ask a question directly to the LLM

print(llm("I want to open a restaurant for Indian food. Suggest a fancy name for it."))

# Output: Maharaja's Palace Cuisine

**Prompt Templates: Smart Fill-in-the-Blanks 📝**

* Instead of constantly changing a full sentence, **Prompt Templates** let you create flexible prompts with placeholders (variables). Think of it like a form you fill out.
* **Code Example (Using Prompt Templates):**

Python

from langchain.prompts import PromptTemplate

from langchain.llms import OpenAI

llm = OpenAI(temperature=0.7)

# Define the prompt template with an input variable 'cuisine'

prompt\_template\_name = PromptTemplate(

input\_variables=['cuisine'],

template="I want to open a restaurant for {cuisine} food. Suggest a fancy name for it."

)

# Format the prompt with a specific cuisine

formatted\_prompt = prompt\_template\_name.format(cuisine="Mexican")

print(formatted\_prompt)

# Output: I want to open a restaurant for Mexican food. Suggest a fancy name for it.

# You can then pass this formatted prompt to the LLM

print(llm(formatted\_prompt))

# Output: Taco Temptation

**Chains: Connecting the Dots 🔗**

* **Chains** are super important in LangChain! They let you link different components (like LLMs and prompt templates) together to create a workflow. It's how you build more complex applications.
  + **LLMChain: The Basic Link**
    - This is the simplest chain. It takes an LLM and a prompt template, and essentially runs them together.
    - **Code Example (LLMChain):**

Python

from langchain.chains import LLMChain

from langchain.prompts import PromptTemplate

from langchain.llms import OpenAI

llm = OpenAI(temperature=0.7)

prompt\_template\_name = PromptTemplate(

input\_variables=['cuisine'],

template="I want to open a restaurant for {cuisine} food. Suggest a fancy name for it."

)

# Create an LLMChain

name\_chain = LLMChain(llm=llm, prompt=prompt\_template\_name)

# Run the chain with your input

print(name\_chain.run("American"))

# Output: The All-American Grille and Bar

* + **Simple Sequential Chain: One Step After Another**
    - This chain is for when you have multiple steps, and the output of one step directly feeds into the input of the next. It's like an assembly line for your AI tasks.
    - **Limitation:** It only gives you the final output, not the intermediate steps.
    - **Code Example (Simple Sequential Chain - concept):**
      * First chain: Generate restaurant name from cuisine.
      * Second chain: Generate menu items from restaurant name.

Python

# (Assuming name\_chain from above is defined)

# Define a prompt template for menu items

prompt\_template\_menu = PromptTemplate(

input\_variables=['restaurant\_name'],

template="Suggest some food menu items for a restaurant called {restaurant\_name}. Return as a comma-separated list."

)

menu\_chain = LLMChain(llm=llm, prompt=prompt\_template\_menu)

from langchain.chains import SimpleSequentialChain

# Create the simple sequential chain

overall\_chain\_simple = SimpleSequentialChain(chains=[name\_chain, menu\_chain]

# Run the chain (input is for the first chain, output is from the last chain)

print(overall\_chain\_simple.run("Indian"))

# Output: (will be a comma-separated list of Indian menu items, not the restaurant name)

* + **Sequential Chain: The Multi-Tasker**
    - This is a more advanced chain. It can handle multiple inputs and produce multiple outputs, making it perfect for complex workflows where you want to see all the results along the way.
    - **Code Example (Sequential Chain - concept):**

Python

from langchain.chains import SequentialChain

# (Assuming name\_chain and menu\_chain from above are defined)

# Important: When using SequentialChain, you need to define 'output\_key'

# for each individual chain so the SequentialChain knows what to expect.

name\_chain = LLMChain(llm=llm, prompt=prompt\_template\_name, output\_key="restaurant\_name")

menu\_chain = LLMChain(llm=llm, prompt=prompt\_template\_menu, output\_key="menu\_items")

# Create the sequential chain

overall\_chain\_sequential = SequentialChain(

chains=[name\_chain, menu\_chain],

input\_variables=['cuisine'], # Input for the first chain

output\_variables=['restaurant\_name', 'menu\_items'] # Outputs you want to see

)

# Run the chain with a dictionary for inputs

response = overall\_chain\_sequential({"cuisine": "Arabic"})

print(response)

# Output: {'cuisine': 'Arabic', 'restaurant\_name': 'The Arabian Bistro', 'menu\_items': 'Hummus, Falafel, Shawarma, Pita Bread'}

**4. Building Your First App: Restaurant Idea Generator 🍽️**

* The video walks through creating a simple web application using **Streamlit** to demonstrate the LangChain concepts. Streamlit is fantastic for data scientists to quickly build user interfaces without needing to dive deep into front-end frameworks like React.

**What to Install for Streamlit:**

Bash

**pip install streamlit**

**Basic Streamlit App Structure (main.py):**

Python

import streamlit as st

import langchain\_helper # This would be your custom file with LangChain logic

st.title("🍽️ Restaurant Name & Menu Generator")

# Create a sidebar for cuisine selection

cuisine = st.sidebar.selectbox(

"Pick a Cuisine",

("Indian", "Mexican", "Italian", "American", "Arabic")

)

if cuisine:

# Call your LangChain helper function (defined in langchain\_helper.py)

response = langchain\_helper.generate\_restaurant\_name\_and\_items(cuisine)

st.header(response['restaurant\_name'].strip()) # .strip() removes extra spaces/newlines

st.write("### Menu Items:")

menu\_items = response['menu\_items'].strip().split(",") # Split into a list

for item in menu\_items:

st.write(f"- {item.strip()}") # Display each item as a bullet point

**LangChain Helper File (langchain\_helper.py):**

* This file encapsulates all your LangChain logic (LLMs, prompt templates, chains).
* **Key components:**
  + Import necessary LangChain classes.
  + Set up your OpenAI(or any you using) API key (as an environment variable or imported from a secret\_key.py file).
  + Define your LLM with a temperature.
  + Create prompt templates for restaurant name and menu items.
  + Construct a SequentialChain to link the name generation and menu item generation.
  + A function generate\_restaurant\_name\_and\_items(cuisine) that takes the cuisine as input and returns the restaurant name and menu items.

**5. Agents: The Smart Decision Makers 🤖**

**What Agents Do**

* Agents are super cool! They use the LLM's **reasoning capabilities** to decide which "tools" to use and in what order to achieve a goal.
* Think of it like this: an LLM isn't just a knowledge base; it's also a smart decision-maker. If you ask ChatGPT Plus about flights, it doesn't just know the answer; it *reasons* that it needs to use an Expedia plugin (a "tool") to find that information.

**Tools: Giving Agents Abilities (Wikipedia, Math, Google Search)**

* Agents have access to **tools** that let them interact with the outside world or perform specific actions.
* **Common Tools:**
  + **Wikipedia Tool:** To search Wikipedia for information (e.g., Elon Musk's birth date).
  + **LLM Math Tool (Calculator):** To perform calculations (e.g., current age based on birth year).
  + **SerpAPI (Google Search Tool):** To perform real-time Google searches (e.g., US GDP in 2022).
    - You need an **API key for SerpAPI** too! Get one from their website after logging in.
    - Set it as an environment variable: os.environ["SERPAPI\_API\_KEY"] = "YOUR\_SERPAPI\_KEY".
* **Code Example (Using Agents with Tools):**

Python

from langchain.llms import OpenAI

from langchain.agents import load\_tools, initialize\_agent, AgentType

import os

# Ensure your OpenAI and SerpAPI keys are set as environment variables

# os.environ["OPENAI\_API\_KEY"] = "..."

# os.environ["SERPAPI\_API\_KEY"] = "..." # For Google Search tool

llm = OpenAI(temperature=0) # Often lower temperature for agents for more predictable reasoning

# Load the tools for the agent

# 'wikipedia' for Wikipedia searches

# 'llm-math' for mathematical operations

# 'serpapi' for Google Search

tools = load\_tools(["wikipedia", "llm-math", "serpapi"], llm=llm)

# Initialize the agent

# AgentType.ZERO\_SHOT\_REACT\_DESCRIPTION is a common type that uses thought and action steps

agent = initialize\_agent(

tools,

llm,

agent=AgentType.ZERO\_SHOT\_REACT\_DESCRIPTION,

verbose=True # Set to True to see the agent's thought process step-by-step!

)

# Run the agent with a complex question

agent.run("When was Elon Musk born and what is his age in 2023?")

# Output will show the agent's thought process, searching Wikipedia, using the calculator, and providing the answer.

**6. Memory: So Your AI Doesn't Forget (Like Us Sometimes! 😉)**

**Why Memory Matters**

* By default, LangChain's chains are "stateless" – they don't remember previous conversations. If you're building a chatbot, this is a big problem! You want it to remember what you just talked about. This is where **Memory** comes in.

**Types of Memory**

* **ConversationBufferMemory: The Full Story**
  + This type of memory stores the **entire transcript** of your conversation. It's great for complete context.
  + **Catch:** If conversations get super long, sending the whole history to the LLM every time can get **expensive** because you're charged per token!
  + **Code Example (ConversationBufferMemory):**

Python

from langchain.memory import ConversationBufferMemory

from langchain.chains import LLMChain

from langchain.prompts import PromptTemplate

from langchain.llms import OpenAI

llm = OpenAI(temperature=0.7)

prompt = PromptTemplate(

input\_variables=["history", "input"],

template="""The following is a friendly conversation between a human and an AI.

Current conversation:

{history}

Human: {input}

AI:"""

)

memory = ConversationBufferMemory(memory\_key="history") # Define memory

conversation\_chain = LLMChain(

llm=llm,

prompt=prompt,

memory=memory # Attach memory to the chain

)

conversation\_chain.run(input="Hi there!")

conversation\_chain.run(input="What is 5 + 5?")

print(memory.buffer) # See the full conversation history

# Output: Human: Hi there! AI: Hello! How can I assist you today? Human: What is 5 + 5? AI: 5 + 5 equals 10.

* **ConversationBufferWindowMemory: Just the Highlights**
  + To avoid the cost issue of **ConversationBufferMemory**, this type of memory only remembers the **last 'k' number of conversational exchanges** (question-answer pairs). It's like having short-term memory, useful for focused conversations.
  + **Code Example (ConversationBufferWindowMemory):**

Python

from langchain.memory import ConversationBufferWindowMemory

from langchain.chains import LLMChain, ConversationChain

from langchain.prompts import PromptTemplate

from langchain.llms import OpenAI

llm = OpenAI(temperature=0.7)

memory = ConversationBufferWindowMemory(k=1) # Remember only the last 1 exchange

# ConversationChain is a convenient wrapper that includes a default prompt and memory

conversation\_chain = ConversationChain(

llm=llm,

memory=memory,

verbose=True # See what's happening internally

)

conversation\_chain.predict(input="Who won the first Cricket World Cup?")

conversation\_chain.predict(input="What is 5 + 5?")

conversation\_chain.predict(input="Who was the captain of the winning team?")

# The last question might get a "I don't know" because it forgot the Cricket World Cup context

# due to k=1, only remembering "What is 5 + 5?".