Introduction to langchain

Hamza Sajid

Why Talk About LangChain?

- 👉*What happens if I give you a dictionary and ask you to have a use it to find meaning of a word?*

- You'll quickly see the problem: the dictionary has knowledge, but it can't:
  - **Remember your last question**
  - **Fetch external information**
  - **Perform calculations or actions**

👉 That's the key idea: **knowledge without connection and memory is limited**.

Library Analogy

- You walk into a massive **library**.
- The books have **all the knowledge**.
- But without a **librarian**, you'll waste time searching, and you might never find the right book.
- An LLM is like a student who memorized a lot of books but doesn't know the latest arrivals or

  your personal notes.

  LangChain is the **librarian assistant** that connects the student to the right books, your notes,

  and even Google for the latest data.

What is Langchain?

- LangChain is an **open-source framework** designed to help developers build applications powered by **Large Language Models (LLMs)**, such as GPT.
- Think of it this way:
- A raw LLM (like GPT) is powerful but **isolated**.
- LangChain acts as a **bridge** that connects LLMs to **data sources, memory, tools, and workflows**.
- In simpler terms:
  👉 If GPT is like a brain, LangChain is like the **nervous system** that allows the brain to interact with the outside world.

Why need Langchain?

- LLMs alone are not enough for **real-world applications**.
- Assume that you have trained an LLM on training data on the question and answer.

- **Is it ready to be used for the production like ChatBot?**

- For just generating the random response it is ready but for creating the production ready app it is not ready to get deployed.

- Because:
- They don’t remember past interactions (no long-term memory).
- They can’t directly access your **private data** or **databases**.
- They struggle with multi-step reasoning tasks that require **tool use** (like search engines, APIs, or calculators).

What LangChain Solves

- LangChain provides:
- **Memory** → Keeps track of past conversations and context.
- **Data Access** → Connects LLMs with external data (documents, databases, APIs).
- **Tool Use** → Allows LLMs to use tools (search engines, Python, SQL).
- **Workflow Management** → Helps you design complex reasoning steps (chains & agents).

The "Memory" Problem: Why Context is King

What is Memory in the Context of LLMs?

Memory allows a chatbot to recall previous parts of a conversation. Without it, the conversation feels disjointed and unnatural.

Real-World Example: A Frustrating Customer Service Chat

Without Memory:

You: "I'd like to know the status of my order."
Chatbot: "Sure, what is your order number?"
You: "It's 12345."
Chatbot: "How can I help you?" (The chatbot has already forgotten the context of your request).

With LangChain's Memory:

LangChain provides various memory modules that can store and recall conversation history. This ensures a continuous and context-aware user experience.

Types of Memory in LangChain: A Deeper Dive

LangChain offers different types of memory to suit various needs:

ConversationBufferMemory:

This is the simplest form, storing the entire conversation history. It's great for short conversations where every detail matters.

ConversationBufferWindowMemory:

To avoid excessively long conversation histories, this type of memory keeps a "window" of the last 'k' interactions.

ConversationSummaryMemory:

For longer conversations, this memory uses an LLM to create a summary of the conversation over time. This is more cost-effective as it uses fewer tokens.

VectorStoreRetrieverMemory:

This advanced memory type stores memories in a VectorDB and retrieves the most relevant information for the current context.

Beyond Memory: Accessing Your Data

A major limitation of LLMs is their inability to access your specific data.

Real-World Example: An Internal Knowledge Base Chatbot

Imagine a chatbot for your company's internal documentation. A standard LLM wouldn't know about your company's specific policies or procedures.

How LangChain Solves This:

Data Loaders: LangChain has over 150 data loaders to ingest data from various sources like PDFs, Word documents, databases, and more.

Retrieval-Augmented Generation (RAG): This is a core feature of LangChain that allows the LLM to retrieve relevant information from your data sources to answer questions. This makes the responses more accurate and contextually relevant.

Unleashing True Power: Tools and Agents

For complex, multi-step tasks, LLMs need access to external tools.

What are Tools and Agents?

Tools are functions that an LLM can use to interact with the outside world, such as a Google Search API, a calculator, or a database query engine.

Agents are the decision-makers. They use the LLM to decide which tools to use and in what order to accomplish a task.

Real-World Example: A Financial Analyst Assistant

Task: "What is the current stock price of Apple, and what are the latest news headlines about the company?"

Without LangChain: An LLM can't answer this as it lacks real-time data access.

With a LangChain Agent:

The agent would first use a **stock price tool** to get the current price.

Then, it would use a **search tool** to find the latest news.

Finally, it would synthesize this information to provide a complete answer.

Why a Framework? The LangChain Advantage

While you could technically build all these functionalities from scratch, a framework like LangChain offers significant advantages:

Abstraction and Modularity: LangChain provides pre-built components for memory, data loading, and tool usage, simplifying the development process.

Standardization: It offers a standard interface for interacting with various LLMs, making it easy to switch between models without rewriting your entire application.

Rapid Prototyping: The modular design allows for quick experimentation and iteration.

Active Community and Integrations: LangChain has a large and active open-source community and integrates with hundreds of third-party tools and services.

Real-World LangChain Use Cases

Intelligent Chatbots: Building context-aware chatbots for customer service, personal assistants, and more.
Question-Answering over Documents: Creating systems that can answer questions based on your private documents and data.
Data Analysis: Automating data analysis tasks by creating agents that can interact with data and perform calculations.
Content Generation: Generating creative and context-aware content like reports, summaries, and marketing copy.
Workflow Automation: Automating complex business processes by creating agents that can interact with various APIs and systems.

LangChain in Action: A Customer Support Chatbot

Let's revisit our customer support chatbot example to see how LangChain brings everything together.

Scenario: A customer wants to know the delivery status of their recent order and also wants to know the return policy for that item.

How the LangChain-powered chatbot handles this:

- **Memory:** The chatbot remembers the customer's order number from the previous turn in the conversation.
- **Tool Use (API Call):** It uses a tool to call the company's internal shipping API to get the real-time delivery status.
- **Data Access (RAG):** It accesses the company's knowledge base to retrieve the specific return policy for the item in the order.
- **LLM for Synthesis:** The LLM then synthesizes all this information into a clear and helpful response for the customer.

Real-World Example

- Imagine you're building a **customer support chatbot**:
- Without LangChain: the chatbot answers only based on what it already knows from training.
- With LangChain:
  - It can fetch product details from your company database.
  - It remembers your last conversation with the customer.
  - It can call external APIs (e.g., shipping tracker) to provide live updates.

👉 Suddenly, the chatbot is **not just answering** but **acting intelligently**.

Where is LangChain Used?

- **Chatbots & Virtual Assistants**
- **Retrieval-Augmented Generation (RAG)** → answering from your private PDFs or databases
- **Automation Agents** → AI that can browse the web, call APIs, or even write & run code
- **Business Intelligence** → querying data with natural language

- LangChain = **Framework that makes LLMs practical** by adding memory, data access, tools, and workflows.

  It is **not a replacement** for GPT or other LLMs, but a **companion** that unlocks their real potential.

Any Question?

Prompting Gen AI models

```python
# import libraries
import os
from dotenv import load_dotenv
from langchain_google_genai import ChatGoogleGenerativeAI

# Load env variables
load_dotenv()
api_key = os.getenv("API_KEY")

# Load Genmini model
llm = ChatGoogleGenerativeAI(
            api_key = api_key,
            model="gemini-1.5-flash",
            temperature=0.0
        )

# Generate the response
response = llm.invoke("What is the Langchain Framework?")
print(response.content)
```

📝 Task for Students

- 👉 Write a Python script that:
- **Initializes the Gemini model** (`gemini-1.5-flash`) with:
  - your API key
  - temperature = `0.0` (deterministic output).
- Provides the model with the following **news article text**:

Tesla shares surged 7% on Monday after the company announced record

deliveries of 466,140 vehicles in the second quarter, surpassing Wall Street

estimates.

Analysts believe strong demand for the Model 3 and Model Y contributed to the increase,

as well as recent price cuts aimed at boosting sales.

- Sends a prompt to the model asking it to:

  “Extract three key bullet points from the following news article.”

- **Prints the summarized response** from the model.

Task for Students

- Text for Extraction Exercise
- Over the past 24 hours, **Bitcoin (BTC)** surged by **5.3%**, crossing the $28,000 mark for the first time in two weeks, fueled by optimism around the upcoming halving event. Analysts pointed out that the **Relative Strength Index (RSI)** is entering overbought territory, signaling short-term caution. Meanwhile, **Ethereum (ETH)** gained **3.1%**, supported by strong demand for staking and a rising number of active addresses. Traders also highlighted the importance of the **50-day moving average** as ETH approaches a key resistance level. On the other hand, **Cardano (ADA)** slipped by **1.8%**, as weaker developer activity raised concerns. However, the **MACD indicator** still shows potential for a bullish reversal in the medium term. Market sentiment overall remains positive, with crypto investors closely monitoring both macroeconomic data and network-level signals to guide their next moves.

Task for Students

- From the above article, extract:
- **Coin names** mentioned.
- **Price change** (positive or negative) for each coin.
- **Indicators** that were discussed.
- Hint: Create a prompt using the f"" string and then generate output in the json format.

Expected output:


```
[
  {"coin": "Bitcoin (BTC)", "price_change": "+5.3%", "indicators": ["Relative Strength Index (RSI)"]},
  {"coin": "Ethereum (ETH)", "price_change": "+3.1%", "indicators": ["50-day moving average"]},
  {"coin": "Cardano (ADA)", "price_change": "-1.8%", "indicators": ["MACD"]}
]
```

Assignment: LLM Extractor + Validator

- Instructions
- In this assignment, you will:
- **Initialize an LLM (Gemini model)** using LangChain.
- **Write an extractor**:
  - Provide the crypto article (given below).
  - Extract **coin name**, **price change**, and **indicators** mentioned.
  - Return the result in **structured JSON format**.
- **Write a validator**:
  - Pass both the **original article text** and the **extracted JSON result** to the LLM.
  - Ask the LLM to check whether the extracted values are **consistent with the article**.
  - The validator should return json format containing
    - `Valid : "Valid"` or `"Invalid"`
    - **Reasoning : Short Reasoning of validation**

- Article Text (use this)

Over the past 24 hours, Bitcoin (BTC) surged by 5.3%, crossing the $28,000 mark

for the first time in two weeks, fueled by optimism around the upcoming halving event.
Analysts pointed out that the Relative Strength Index (RSI) is entering overbought territory, signaling

short-term caution. Meanwhile, Ethereum (ETH) gained 3.1%, supported by strong demand for

staking and a rising number of active addresses. Traders also highlighted the importance

of the 50-day moving average as ETH approaches a key resistance level. On the other hand,

Cardano (ADA) slipped by 1.8%, as weaker developer activity raised concerns. However, the MACD

indicator still shows potential for a bullish reversal in the medium term. Market sentiment

overall remains positive, with crypto investors closely monitoring both macroeconomic data

and network-level signals to guide their next moves.

Thank You