LangChain

1️⃣ Introduction

LangChain is an **open-source framework** that helps developers build applications using **Large Language Models (LLMs)** like OpenAI's GPT, Anthropic's Claude, and others.
While LLMs are great at generating text or understanding prompts, LangChain provides the **infrastructure to connect LLMs with external data, tools, and multi-step logic**, enabling the creation of complex and dynamic applications.

LangChain is particularly useful for:

- Building LLM-powered **chatbots**
- Creating **AI agents** that interact with tools like search engines or APIs
- Querying **databases using natural language**
- Working with **documents, PDFs, and more**

2️⃣ Core Concepts / Components

1. **LLMs and Chat Models**
   LangChain supports various LLMs and chat models. These models generate responses based on user prompts.

Example:

```
from langchain.llms import OpenAI

llm = OpenAI(model_name="gpt-4")

response = llm("What is the capital of France?")

print(response)
```

Output:

```
The capital of France is Paris.
```

2. **Prompt Templates**
   Allows dynamic prompts with placeholders for variables.

Example:

```
from langchain.prompts import PromptTemplate

template = PromptTemplate.from_template("What is the capital of {country}?")
```

```
prompt = template.format(country="Germany")

print(prompt)
```

Output:

```
What is the capital of Germany?
```

3. **Chains**
   Chains combine prompts and LLMs into a workflow for sequential processing.

Example:

```
from langchain.chains import LLMChain

chain = LLMChain(llm=llm, prompt=template)

result = chain.run("Japan")

print(result)
```

Output:

```
The capital of Japan is Tokyo.
```

4. **Agents and Tools**
   Agents are autonomous LLM-powered assistants that choose which tool to use based on context.

Tools include:

- Search (Google, SerpAPI)
- Calculator (math operations)
- API access

Example scenario:
A customer support AI that:

4. Searches the FAQ for answers
5. Accesses the database for additional info
6. Responds conversationally via the LLM

3️⃣ Use Cases / Applications

- **Document Q&A Systems** – Upload PDFs or Word docs and ask questions like "Summarize section 3."

- **Intelligent Customer Support** – Integrate with CRM and knowledge bases to respond to user queries.
- **SQL Database Querying** – Convert natural language into SQL queries: e.g., “Show all customers with purchases over $500 last month.”
- **AI Agents / Copilots** – Assist users with tasks like booking tickets, fetching data, or coding autonomously.

4️⃣ Related Tools / Integrations

LangChain integrates with:

- LLM providers (OpenAI, Anthropic, Cohere)
- Databases and vector stores (FAISS, Pinecone) for RAG
- APIs and custom tools for agent execution

Example: Multi-step chain integrating a database search with LLM response.

5️⃣ Conclusion

- LangChain provides the **infrastructure** to build advanced LLM-powered applications.
- It supports **dynamic workflows, memory, agents, and integration** with external tools.
- Enables developers to build **intelligent, autonomous, and scalable AI systems**.

PromptLayer

1️⃣ Introduction

PromptLayer is a **prompt engineering and observability platform** that helps developers **track, manage, and debug prompts** used with Large Language Models (LLMs).
Think of it as an **analytics dashboard + version control system** for your AI prompts.

PromptLayer is particularly useful for:

- Monitoring **prompt performance**
- Comparing outputs from different prompts or models
- Debugging errors and inconsistencies
- Collaborating on prompt experiments
- Optimizing token usage and cost

2️⃣ Core Concepts / Components

1. **Prompt Logging**
   Automatically records every prompt and its response along with metadata.

Example:

```
import promptlayer

from langchain.llms import OpenAI


llm = OpenAI(openai_api_key="YOUR_KEY", pl_tags=["customer-support-bot"])

response = llm("Explain the return policy.")

print(response)
```

Output:

```
Our return policy allows customers to return items within 30 days of purchase.
```

You can now see this prompt, response, token usage, and execution time in PromptLayer's dashboard.

2. **Prompt Version Control**
   Create and test multiple versions of the same prompt.

Example:

   - **V1:** "Summarize this article."
   - **V2:** "Summarize the article in bullet points for a 10-year-old."
     PromptLayer allows comparing performance and outputs of each version.

3. **Prompt Comparison & A/B Testing**
   Analyze metrics like:
   - Accuracy of response

    - Token usage and cost
    - Response time
      This is helpful to optimize prompts in production systems.

4. **Team Collaboration**
    - Share prompt experiments and logs with team members
    - Organize workspaces for developers, data scientists, and product teams

---

3️⃣ Use Cases / Applications

- **Debugging LLM Applications:** Track which prompts caused failures or bad responses.
- **Cost Optimization:** Identify expensive prompts and reduce token usage.
- **Scaling Prompt Engineering:** Manage thousands of prompts with tags, filters, and version control.
- **Production Monitoring:** Ensure consistent performance of prompts in chatbots, assistants, and AI agents.

4️⃣ Related Tools / Integrations

PromptLayer integrates seamlessly with:

- **LangChain** – Logs every prompt used in chains, agents, and workflows
- **OpenAI API** – Tracks usage, tokens, and model responses
- **Team dashboards** – Collaborative prompt management and analysis

Example: Using PromptLayer with LangChain

from langchain.llms import OpenAI

from langchain.prompts import PromptTemplate

from langchain.chains import LLMChain

from langchain_community.callbacks.promptlayer import PromptLayerCallbackHandler

callback_handler = PromptLayerCallbackHandler()

```
llm = OpenAI(model="gpt-3.5-turbo", callbacks=[callback_handler])

template = PromptTemplate.from_template("Summarize this text: {text}")

chain = LLMChain(llm=llm, prompt=template)


response = chain.run({"text": "LangChain is a framework for building LLM-powered apps."})

print(response)
```

5️⃣ Conclusion

- PromptLayer is a **tool for tracking, debugging, and optimizing prompts**.
- Combined with LangChain, it allows developers to **build reliable, maintainable, and intelligent AI applications**.
- Provides **transparency, reproducibility, and collaborative workflow management** for prompt engineering.