DAY – 10

PART 1: LANGCHAIN AND AGENTS

On Day 10, we explored the architecture and components of LangChain, a Python-based framework designed to enable the development of advanced AI-powered applications. LangChain empowers developers to build intelligent agents that can interact with tools, maintain memory, and handle multi-turn conversations using LLMs (Large Language Models).

1. WHAT IS LANGCHAIN?

LangChain is an open-source framework that simplifies the process of creating AI applications that use language models to perform dynamic, multi-step tasks. It allows for the combination of different components such as prompts, chains, memory, and tools into coherent workflows.

2. KEY STRENGTHS

LangChain offers three main strengths:

- Orchestration: Connects LLMs, tools, and memory to execute complex workflows.
- Modularity: Each component (like chains, prompts, tools) can be designed independently.
- Agent Support: Enables dynamic decision-making and real-time interaction with tools and APIs.

3. CORE CONCEPTS

We studied the five essential components:

- **LLM:** The language model powering generation.
- **Chain:** A sequence of operations, often combining prompts and logic.
- **Tool:** An external function the model can call (e.g., calculator, web search).
- **Memory:** Stores context from previous interactions to enable multi-turn conversations.
- **Agent:** A smart system that decides, in real-time, which tools and responses to use.

4. LANGCHAIN SETUP

We went through the setup process, which includes:

- Installing LangChain via pip.
- Setting up environment variables for API keys.
- Importing required modules and configuring LLMs.

5. CHAINS IN LANGCHAIN

We built simple chains to see how different prompts can be linked together to solve tasks. This included:

- Sequential chains (A → B → C)
- Summarization + Q&A
- Query + calculation + summarization

6. TOOLS – EXTENDING LLMs

LangChain allows LLMs to interact with tools such as:

- Python REPL
- Web Search APIs
- Calculator
- Document loaders

These tools extend the LLM's functionality beyond static text generation.

7. AGENTS & 8. USE CASES

Agents can:

- Make decisions step-by-step
- Choose which tool to use based on context
- Handle long-form conversations

Use cases include:

- Conversational assistants
- Automated data retrieval

- Task-based agents (e.g., booking travel, summarizing web pages)

9. BUILDING AN AGENT WITH TOOLS

We built a basic agent using:

- Gemini LLM
- Calculator and search tools
- Prompt templates

The agent was capable of interpreting input, selecting tools, and returning actionable results.

10. MULTI-TURN CONVERSATION

We used memory components to retain conversation history, enabling the agent to:

- Remember user names
- Follow-up on prior topics
- Maintain conversational continuity

11. ADVANCED AGENT PATTERNS

We explored strategies like:

- ReAct (Reasoning + Action)
- Conversational ReAct
- Tool-Calling Patterns

These allow for smarter, more responsive agent behavior.

12. HOW WE CAN DO THIS

We implemented LangChain components step-by-step:

- Set up the environment
- Created prompt templates
- Built chains
- Added tools

- Wrapped everything into an AgentExecutor

PART 2: PROMPT EVALUATION TECHNIQUES

13. WHAT IS PROMPT EVALUATION?

Prompt evaluation is the process of analyzing the quality, performance, and reliability of prompts used in LLM workflows. This ensures better results, efficiency, and usability in production systems.

14. MAIN PROMPT EVALUATION TECHNIQUES

We studied several key techniques:

- **Human Evaluation:** Manually rating the quality and clarity of the output.
- **Automatic Evaluation:** Using models or scoring functions to auto-grade responses.
- **Metric-Based Evaluation:** BLEU, ROUGE, BERTScore to compare outputs against references.
- **A/B Testing:** Comparing prompt variants to determine better performance.
- Few-Shot vs Zero-Shot Prompt Comparison
- **Prompt Robustness Testing:** Changing phrasing or tone slightly to see performance variation.
- **Prompt Sensitivity Testing:** Altering structure or input order.
- **Response Diversity Testing:** Checking variability in model responses to the same prompt.
- **Cost and Speed Testing:** Measuring efficiency based on token usage and latency.

15. CODE IMPLEMENTATION USING GOOGLE AI STUDIO API

We implemented prompt evaluation techniques using Google AI Studio and Gemini 1.5 by:

- Writing Python scripts to call different prompt variations.
- Logging model output, response time, and token usage.
- Comparing side-by-side outputs using A/B testing logic.
- Automating evaluation using scoring scripts and system prompts.

CONCLUSION

Day 10 was a deeply technical and hands-on session, where we covered both the LangChain framework for intelligent agents and the essential evaluation techniques required to build and maintain high-quality AI systems. We learned how to create intelligent, multi-tool agents and evaluate their prompt logic with a combination of human and automated methods using Gemini and Google AI Studio.