Training Day 11 Report:

Overview

LangChain supports **function-based tool use** in agents, allowing LLMs to call external tools and APIs during reasoning. This feature enables dynamic, real-time interactions where a language model doesn't just generate text but can also fetch data, perform calculations, or trigger workflows.

What Are Functions in LangChain?

Functions (or tools) in LangChain are Python-callable modules that can be invoked by an LLM when it determines a tool is required. These tools can include:

- API wrappers (e.g., weather, currency, Wikipedia)
- Database queries
- Math solvers or code runners

The LLM outputs a JSON object describing the function to call, and LangChain executes it and returns the result to the model for a follow-up response.

Example: API Integration

Use Case: Fetching weather using a public API.

1. Define a function:
    - Name: `get_weather(city)`
    - Action: Sends a request to a weather API using `requests.get()`.

2. Register the function in LangChain using the `Tool()` constructor.
3. Add the tool to an agent's toolset.
4. LLM chooses to call `get_weather` when prompted with "What's the weather in Delhi?"

Tool Execution Flow

1. User prompt: `"Find the current temperature in Mumbai."`
2. LLM decides: Use tool `get_weather`.
3. LangChain executes the function via API.
4. Result (e.g., "32°C and clear") is returned to the LLM.
5. LLM replies: "It's currently 32°C and clear in Mumbai."

LangChain Functions vs Traditional Prompts

With Functions/Tools	**Without Tools**
Can fetch live data (e.g., time, weather)	Limited to model's training cutoff
Structured interaction between model and logic	Relies on plain text generation only
Useful for agents, automation	More suited for static Q&A or summarization

Common API Use Cases

- **Weather or news lookup**
- **Stock prices and currency conversion**
- **Wikipedia or web search (DDGS)**
- **Math or code evaluation (Python REPL)**

Conclusion

LangChain's function calling and API tool support make LLM applications more powerful, interactive, and context-aware. By bridging the gap between static language generation and real-time data access, developers can build AI agents capable of performing actions, looking up information, and making informed decisions based on up-to-date knowledge.