Tools in LangChain

1. Introduction

In **LangChain**, a **Tool** is a Python function or API packaged in a way that an LLM can understand and call when needed. Although LLMs are powerful in reasoning and language generation, they cannot directly access real-time data, run code, or interact with external systems. **Tools bridge this gap**, enabling LLMs to take real-world actions and perform useful operations beyond text generation.

Definition

A **Tool** is a callable Python function that allows an LLM to perform a specific action — such as searching the web, querying a database, or sending a message.

2. Capabilities of LLMs

LLMs Can Do	LLMs Cannot Do
• Reasoning • Natural Language Understanding • Contextual Response Generation	• Access live or external data • Perform complex computations reliably • Call APIs or execute system commands • Directly interact with files or databases

3. Types of Tools in LangChain

3.1. Built-in Tools

Built-in tools are **ready-to-use utilities** provided by LangChain. They require no custom logic — just import and use them.

Common Built-in Tools

- `DuckDuckGoSearchRun` — Performs real-time web search.
- `WikipediaQueryRun` — Retrieves information from Wikipedia.
- `PythonREPLTool` — Executes Python code in a REPL environment.
- `ShellTool` — Runs shell or terminal commands.
- `RequestsGetTool` — Sends HTTP GET requests.
- `SQLDatabaseQueryTool` — Executes SQL queries.
- `GmailSendMessageTool`, `SlackSendMessageTool` — Sends messages via integrations.

3.2. Custom Tools

Custom tools are user-defined functions that extend LangChain's ability to work with private APIs, internal systems, or application logic.

Use them when:

- Integrating your own API or backend service.
- Implementing organization-specific logic.
- Allowing the LLM to interact with your product.

4. Creating Custom Tools

LangChain provides multiple ways to define tools based on the complexity of the task.

Methods to Create Custom Tools

- Using the **@tool** decorator — for simple callable functions.
- Using **StructuredTool** with **Pydantic** — for argument validation.
- Extending the **BaseTool** class — for fully customized logic.

4.1. Structured Tool

A **StructuredTool** uses a **Pydantic** schema to ensure structured and validated input/output from the LLM.

4.2. BaseTool Class

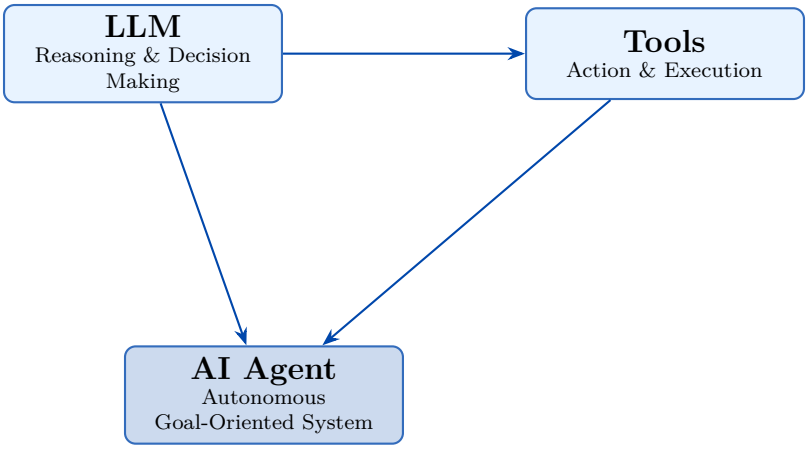
The **BaseTool** class defines the foundation for all tools. It manages how a tool executes, handles input/output, and integrates with the broader LangChain ecosystem.

- Defines core execution interface.
- Enables full control and customization.

5. How Tools Fit into the Agent Ecosystem

An AI Agent combines:

- **LLM:** Handles reasoning, planning, and decision-making.
- **Tools:** Perform real-world actions and data retrieval.



This interaction allows agents to *think* (LLM) and *act* (Tools) — forming the foundation of intelligent, goal-driven systems.

6. Toolkits in LangChain

A **Toolkit** is a modular collection of related tools designed for reuse and rapid deployment.

Example: Google Drive Toolkit

GoogleDriveToolkit may include:

- `GoogleDriveCreateFileTool` — Create new files.
- `GoogleDriveSearchTool` — Search existing files.
- `GoogleDriveReadFileTool` — Retrieve file contents.

Toolkits simplify development and improve scalability in agent-based systems.

7. Summary

- Tools extend LLM functionality beyond language reasoning.
- Built-in tools handle common API and system tasks.
- Custom tools integrate domain-specific logic.
- Agents combine LLM reasoning with tool execution.
- Toolkits offer modular, reusable sets of tools.