Reading: Tools, Agents, and Function Calling in LangChain

Estimated time: 9 mins

Learning Objectives

By the end of this reading, you will be able to:

- Describe the key components and workflow of tool calling in LangChain, including how LLMs generate structured tool calls using parameters and descriptions.
- Differentiate between tool calling and function calling, and explain how agents utilize tools, memory, and actions to interact with external systems.

This reading explores how large language models (LLMs) interact with external tools through structured "tool calling" or "function calling." It explains key components of tools in LangChain, how agents make decisions using tools and memory, and how structured tool invocation enables dynamic, real-time interactions with the external world to solve complex tasks. Let's start by understanding what tools are in LangChain.

Tools

Tools are essentially functions made available to LLMs. For example, a weather tool could be a Python or a JavaScript function with parameters and a description that fetches the current weather of a location.

A tool in LangChain has several important components that form its schema:

- **Name:** A unique identifier for the tool.
- **Description:** A brief explanation of the tool's purpose.
- **Parameters:** The inputs the tool expects to function correctly.

This schema enables the LLM to understand when and how to use the tool effectively.

Tool calling

Contrary to the term, in tool calling, LLMs do not execute tools or functions directly. Instead, they generate a structured representation indicating which tool to use and with what parameters.

When you pose a question to the LLM that requires external information or computation, the model evaluates the available tools based on their names and descriptions. If it identifies a relevant tool, the model generates a structured output (typically formatted as a JSON object) that specifies the tool's name and appropriate parameter values. This is still text generation, just in a structured format intended for tool input.

An external system then interprets this structured output, executes the actual function or API call, and retrieves the result. This result is subsequently fed back to the LLM, which uses it to generate a comprehensive response.

Here's the workflow example in simple terms:

- Define a weather tool and ask a question like: "What's the weather like in NY?"
- The model halts regular text generation and outputs a structured tool call with parameter values (e.g., "location": "NY").
- Extract the tool input, execute the actual weather-checking function, and obtain the weather details.
- Pass the output back to the model so it can generate a complete final answer using the real-time data.

Function calling vs. tool calling

Function calling and tool calling refer to essentially the same concept, with different terminology:

- **Function calling** is the term popularized by OpenAI in their API documentation
- **Tool calling** is a more general industry term used by multiple AI providers, including Anthropic, and broadly in frameworks like LangChain
- Both describe the same capability: enabling an LLM to request specific external functions to be executed with structured parameters

The concepts, workflows, and implementations are functionally identical - the difference is primarily in naming convention rather than technical distinction.

Tools in LangChain

Tools are utilities designed to be called by a model: their inputs are structured in a way that models can generate, and their outputs are intended to be passed back to the model. These tools perform specific actions such as searching the web, querying databases, or executing code.

A **toolkit** is a collection of related tools designed to work together for a common purpose or integration.

Ways to initialize and use tools

LangChain provides several methods to initialize and use tools:

1. **Using Built-in Tools:** LangChain offers a variety of built-in tools for common tasks. For example, the WikipediaQueryRun tool allows fetching data from Wikipedia.
2. **Loading Tools with load_tools:** LangChain provides a load_tools function to load multiple tools conveniently. This function can be used to load tools like wikipedia, serpapi, llm-math, etc.
3. **Creating Custom Tools:** You can define your own tools using the Tool class or the @tool decorator. This allows you to wrap any function as a tool that the LLM can invoke.
4. **Tools as OpenAI Functions:** LangChain tools can be converted to OpenAI functions using the convert_to_openai_function utility. This allows you to use LangChain tools with OpenAI's function calling API. Additionally, you can use bind_functions or bind_tools methods to automatically convert and bind tools to your OpenAI chat model.

LangChain supports a wide range of external tools bundled into toolkits. These allow LLMs to interact with real-world data sources, APIs, and services. Some popular categories include:

- **Wikipedia:** Fetch summaries and information from Wikipedia articles.
- **Search engines:** Integrate with search engines like Bing, Google, and DuckDuckGo to fetch real-time search results.
- **APIs:** Access various APIs for tasks like weather data retrieval, financial data analysis, etc.

and many more!

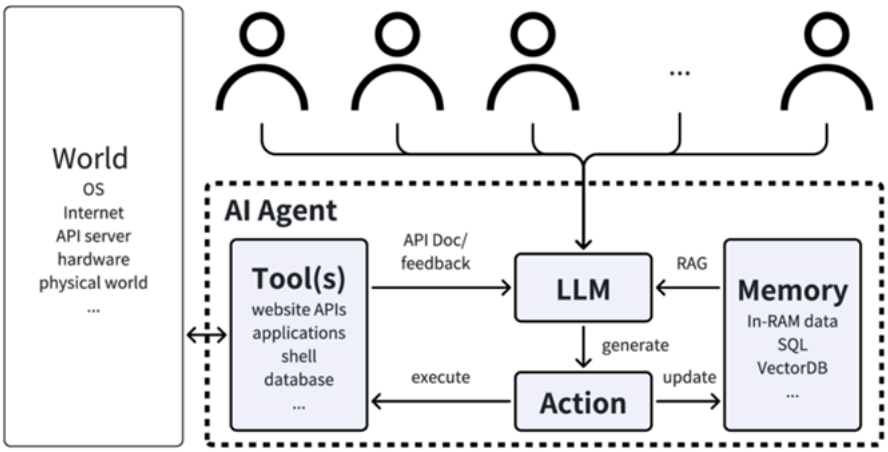
Agents

Agents are decision-making systems powered by LLMs that can reason, use tools, access memory, and take actions to complete tasks. Unlike a standalone LLM that only generates text, an agent figures out how to respond, which might involve searching for information, querying a database, or executing code.

An **agent** is a high-level orchestration system rather than a single function (like tool); it encapsulates the LLM itself along with supporting components like tools, memory, and an execution framework. In essence, tool calling allows agents to leverage tools to interact with the real world, making them capable of dynamic, context-aware decision-making and real-world problem solving beyond text generation alone.

Architecture of an AI agent in LangChain

The following image illustrates the architecture of an **AI Agent** in LangChain, a system where an LLM is empowered to make decisions, use tools, access memory, and interact with the external world.



Reference: **Overview-of-LLM-based-AI-agent**

Let's discuss all the components in the architecture of an AI agent in LangChain.

1. **AI Agent**
   This is the main intelligence unit. It includes LLM, tools, memory, and logic to take actions. The agent is responsible for processing input queries and figuring out how to respond, not just with text, but by taking actions when needed.

2. **Large Language Model (LLMs)**
   At the core of the agent, the LLM interprets the input and determines what to do next. It may decide to:

- Recall something from memory (via Retrieval-Augmented Generation or "RAG")
- Use a tool
- Directly generate a response

3. **Tool(s)**
   As explained earlier, they are functions or external capabilities that an agent can use. The LLM generates structured requests specifying which tool to use and with what parameters. The tool is then executed outside the model, and the result is returned.

4. **Memory**
   Memory allows the agent to store and retrieve useful information, enabling long-term context and personalized interaction. It could include:

- Short-term in-RAM data
- Structured storage like SQL
- Semantic memory in VectorDBs (e.g., for document search)

5. **Action**
   When the LLM decides a tool needs to be used, it outputs a structured "action," which is then executed by the system. The outcome of this action is either shared with the user or used in further reasoning steps.

6. **Connection to the external world**
   The World represents everything outside the AI agent — the OS, internet, APIs, physical devices, etc. The agent can interact with these systems through tools, bridging the gap between the LLM and real-world applications.

Example

Let's see an example of the flow:

1. **User asks:**

- *"What is the weather in New York?"*
- The input is received by the AI agent. This is a natural language query requiring real-time information that the LLM alone may not have.

2. **LLM processes the query and identifies the need for a tool:**

- The LLM understands that it doesn't have real-time weather data and determines that a **weather tool** (e.g., an API) should be used. It prepares a **structured tool call**, specifying the tool name and input parameters (e.g., location = "New York").

3. **Tool is invoked as an action:**

- The agent passes this structured request to the external weather tool. The tool is executed, reaching out to a weather API to fetch the latest conditions for New York. This is the "action" step, where the LLM's plan is turned into a real-world operation.

4. **Memory may be referenced or updated:**

- If the user has asked about weather before, the agent might reference this prior context from **memory** to personalize or compare results. Memory can also be updated with the new interaction for future reference.

5. **LLM receives tool output and generates a response:**

- Once the tool returns the result (e.g., "It's currently 68°F and sunny"), the LLM takes this output and uses it to generate a complete, user-friendly response.

6. **Agent responds and completes the task using real-world data:**

- The user receives the final output:
  - *"It's currently 68 degrees and sunny in New York City."*
  - The task is now completed using both the LLM's processing capability and external world data accessed via the tool.

Agents in LangChain

LangChain offers several methods to create and utilize agents:

1. **Using built-in agent types**
   LangChain offers predefined agent types such as zero-shot-react-description and chat-zero-shot-react-description. These are useful for simple reasoning tasks where the model decides which tool to use based on the description alone.

2. **Creating agents with OpenAI functions**
   LangChain supports agents that can leverage OpenAI's function-calling capabilities. You can create such agents using create_openai_functions_agent, which makes it easy to interact with structured tools in a safe and predictable way.

3. **Building agents with LangGraph**
   LangGraph is a low-level orchestration library that provides more flexibility and control. Using the high-level create_react_agent method, you can create agents capable of complex reasoning, action chaining, memory integration, and real-time streaming.

4. **Executing agents with AgentExecutor**
   Agents are executed using AgentExecutor, which manages the loop of calling the LLM, processing tool outputs, and determining when the final response is ready. This ensures the agent can dynamically react to intermediate results during its reasoning process.

5. **Adding memory**
   By using components like MemorySaver, agents can maintain context across multiple interactions. This allows for personalized and context-aware responses, where an agent can remember things like user preferences, past questions, or conversation history.

Summary

In this reading, you learned about:

1. **Tool calling fundamentals:** Tool calling allows LLMs to generate structured requests (not actual execution) that specify which external tool to use and with what parameters, enabling access to real-time or external data.
2. **Components of a tool:** A tool in LangChain includes a **name**, **description**, and **parameters**. This schema helps the model understand how and when to use each tool.
3. **Agents vs. tools:** Agents are advanced systems powered by LLMs that can make decisions, call tools, access memory, and take actions, enabling dynamic task completion beyond text generation.
4. **Workflow of tool invocation:** When a question requires external data, the LLM outputs a structured tool call. An external system executes the tool, retrieves results, and feeds them back to the LLM for a complete answer.
5. **Function calling vs. tool calling:** These terms are technically identical. "Function calling" is OpenAI's terminology; "tool calling" is used more broadly in LangChain and other frameworks.

Author(s)

IBM Skills Network Team