阅读：LangChain中的工具、代理和函数调用

预计时间： 9分钟

学习目标

通过本次阅读，您将能够：

- 描述 LangChain 中工具调用的关键组成部分和工作流程，包括 LLM 如何使用参数和描述生成结构化工具调用。
- 区分工具调用和函数调用，并解释代理如何利用工具、记忆和动作与外部系统进行交互。

本次阅读探讨了大型语言模型（LLMs）如何通过结构化的“工具调用”或“函数调用”与外部工具进行交互。它解释了 LangChain 中工具的关键组成部分，代理如何利用工具和记忆做出决策，以及结构化工具调用如何实现与外部世界的动态实时交互，以解决复杂任务。让我们先来了解 LangChain 中的工具是什么。

工具

工具本质上是提供给大型语言模型（LLMs）的功能。例如，天气工具可以是一个Python或JavaScript函数，带有参数和描述，用于获取某个地点的当前天气。

LangChain中的工具有几个重要组件，构成其架构：

- **名称：** 工具的唯一标识符。
- **描述：** 对工具目的的简要说明。
- **参数：** 工具正常运行所需的输入。

这个架构使得LLM能够理解何时以及如何有效地使用工具。

工具调用

与术语相反，在工具调用中，LLM 并不直接执行工具或函数。相反，它们生成一个结构化的表示，指示使用哪个工具以及使用什么参数。

当你向 LLM 提出一个需要外部信息或计算的问题时，模型会根据可用工具的名称和描述来评估这些工具。如果它识别出一个相关的工具，模型会生成一个结构化输出（通常格式为 JSON 对象），指定工具的名称和适当的参数值。这仍然是文本生成，只是以一种结构化的格式，旨在用于工具输入。

然后，外部系统会解释这个结构化输出，执行实际的函数或 API 调用，并检索结果。这个结果随后被反馈给 LLM，LLM 使用它生成一个全面的响应。

以下是简单术语的工作流程示例：

- 定义一个天气工具，并提出一个问题，比如：“纽约的天气怎么样？”
- 模型停止常规文本生成，并输出一个带有参数值的结构化工具调用（例如，“location”：“NY”）。
- 提取工具输入，执行实际的天气检查函数，并获取天气详情。
- 将输出反馈给模型，以便它可以使用实时数据生成完整的最终答案。

函数调用与工具调用

函数调用和工具调用本质上指的是相同的概念，只是术语不同：

- **函数调用** 是OpenAI在其API文档中推广的术语
- **工具调用** 是多个AI提供商（包括Anthropic）使用的更通用的行业术语，并且在像LangChain这样的框架中广泛使用
- 两者描述的是相同的能力：使LLM能够请求执行特定的外部函数，并带有结构化参数

这些概念、工作流程和实现功能上是相同的——区别主要在于命名约定，而不是技术上的区别。

LangChain中的工具

工具是旨在被模型调用的实用程序：它们的输入以模型可以生成的方式进行结构化，输出则旨在传回模型。这些工具执行特定的操作，例如搜索网络、查询数据库或执行代码。

工具包是一组相关工具的集合，旨在为共同的目的或集成而协同工作。

初始化和使用工具的方法

LangChain 提供了几种初始化和使用工具的方法：

1. **使用内置工具：** LangChain 提供了多种内置工具用于常见任务。例如，WikipediaQueryRun 工具允许从维基百科获取数据。
2. **使用 load_tools 加载工具：** LangChain 提供了一个 load_tools 函数来方便地加载多个工具。此函数可用于加载像 wikipedia、serpapi、llm-math 等工具。
3. **创建自定义工具：** 您可以使用 Tool 类或 @tool 装饰器定义自己的工具。这允许您将任何函数包装为 LLM 可以调用的工具。
4. **将工具作为 OpenAI 函数：** LangChain 工具可以使用 convert_to_openai_function 工具转换为 OpenAI 函数。这使您能够将 LangChain 工具与 OpenAI 的函数调用 API 一起使用。此外，您还可以使用 bind_functions 或 bind_tools 方法自动转换并绑定工具到您的 OpenAI 聊天模型。

LangChain 支持广泛的外部工具，打包成工具包。这些工具允许 LLM 与现实世界的数据源、API 和服务进行交互。一些流行的类别包括：

- **维基百科：** 从维基百科文章中获取摘要和信息。
- **搜索引擎：** 与 Bing、Google 和 DuckDuckGo 等搜索引擎集成，以获取实时搜索结果。
- **API：** 访问各种 API，进行天气数据检索、金融数据分析等任务。

还有更多！

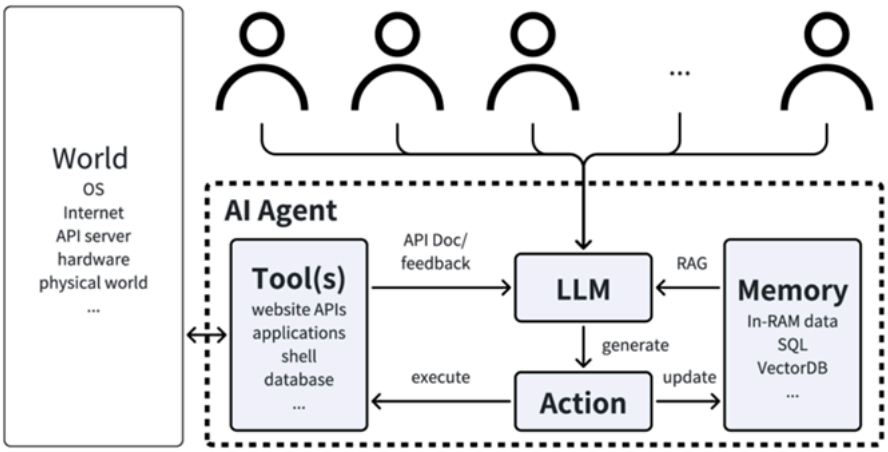
代理

代理是由 LLM 驱动的决策系统，能够推理、使用工具、访问记忆并采取行动以完成任务。与仅生成文本的独立 LLM 不同，代理会找出如何回应，这可能涉及搜索信息、查询数据库或执行代码。

一个 **代理** 是一个高层次的编排系统，而不是单一功能（如工具）；它封装了 LLM 本身以及支持组件，如工具、记忆和执行框架。本质上，工具调用使代理能够利用工具与现实世界互动，使它们能够进行动态、上下文感知的决策和超越文本生成的现实问题解决。

LangChain中AI代理的架构

以下图像展示了LangChain中**AI代理**的架构，这是一个使LLM能够做出决策、使用工具、访问记忆并与外部世界互动的系统。



参考： [基于LLM的AI代理概述]

让我们讨论LangChain中AI代理架构的所有组件。

1. **AI代理**
   这是主要的智能单元。它包括LLM、工具、记忆和采取行动的逻辑。代理负责处理输入查询并确定如何响应，不仅仅是通过文本，而是在需要时采取行动。

2. **大型语言模型（LLMs）**
   在代理的核心，LLM解释输入并确定接下来要做什么。它可能决定：

- 从记忆中回忆某些内容（通过检索增强生成或“RAG”）
- 使用工具
- 直接生成响应

3. **工具**
   如前所述，它们是代理可以使用的函数或外部能力。LLM生成结构化请求，指定使用哪个工具以及使用什么参数。然后在模型外部执行该工具，并返回结果。

4. **记忆**
   记忆允许代理存储和检索有用的信息，从而实现长期上下文和个性化互动。它可以包括：

- 短期内存数据
- 结构化存储，如SQL
- 在VectorDB中的语义记忆（例如，用于文档搜索）

5. **行动**
   当LLM决定需要使用工具时，它输出一个结构化的“行动”，然后由系统执行。此行动的结果要么与用户共享，要么用于进一步的推理步骤。

6. **与外部世界的连接**
   世界代表AI代理之外的所有事物——操作系统、互联网、API、物理设备等。代理可以通过工具与这些系统互动，架起LLM与现实世界应用之间的桥梁。

示例

让我们来看一个流程示例：

1. **用户提问：**

- *“纽约的天气怎么样？”*
- 输入被AI代理接收。这是一个自然语言查询，需要实时信息，而LLM本身可能没有。

2. **LLM处理查询并识别工具需求：**

- LLM理解到它没有实时天气数据，并确定应该使用一个**天气工具**（例如，一个API）。它准备了一个**结构化工具调用**，指定工具名称和输入参数（例如，位置 = “纽约”）。

3. **工具作为一个动作被调用：**

- 代理将这个结构化请求传递给外部天气工具。该工具被执行，联系一个天气API以获取纽约的最新天气情况。这是“动作”步骤，LLM的计划转化为实际操作。

4. **可能引用或更新记忆：**

- 如果用户之前询问过天气，代理可能会引用**记忆**中的先前上下文，以个性化或比较结果。记忆也可以随着新的交互进行更新，以供将来参考。

5. **LLM接收工具输出并生成响应：**

- 一旦工具返回结果（例如，“当前温度为68°F，天气晴朗”），LLM将此输出用于生成完整的、用户友好的响应。

6. **代理使用实时数据回应并完成任务：**

- 用户收到最终输出：
  - *“纽约市当前温度为68度，天气晴朗。”*
  - 任务现在已经使用LLM的处理能力和通过工具访问的外部世界数据完成。

LangChain中的代理

LangChain提供了几种创建和使用代理的方法：

1. **使用内置代理类型**
   LangChain提供了预定义的代理类型，如zero-shot-react-description和chat-zero-shot-react-description。这些对于简单的推理任务非常有用，模型仅根据描述决定使用哪个工具。

2. **使用OpenAI函数创建代理**
   LangChain支持可以利用OpenAI函数调用能力的代理。您可以使用create_openai_functions_agent创建这样的代理，这使得与结构化工具的交互变得安全且可预测。

3. **使用LangGraph构建代理**
   LangGraph是一个低级的编排库，提供了更多的灵活性和控制。通过使用高级的create_react_agent方法，您可以创建能够进行复杂推理、动作链、内存集成和实时流式传输的代理。

4. **使用AgentExecutor执行代理**
   代理通过AgentExecutor执行，该执行器管理调用LLM的循环、处理工具输出，并确定最终响应何时准备好。这确保代理能够在推理过程中动态响应中间结果。

5. **添加记忆**
   通过使用MemorySaver等组件，代理可以在多次交互中保持上下文。这允许个性化和上下文感知的响应，代理可以记住用户偏好、过去的问题或对话历史等信息。

摘要

在本次阅读中，您了解了：

1. **工具调用基础：** 工具调用允许大型语言模型生成结构化请求（而非实际执行），指定使用哪个外部工具及其参数，从而能够访问实时或外部数据。
2. **工具的组成部分：** LangChain 中的工具包括 **名称**、**描述** 和 **参数**。该模式帮助模型理解如何以及何时使用每个工具。
3. **代理与工具：** 代理是由大型语言模型驱动的高级系统，可以做出决策、调用工具、访问记忆并采取行动，从而实现超越文本生成的动态任务完成。
4. **工具调用的工作流程：** 当问题需要外部数据时，大型语言模型输出一个结构化的工具调用。外部系统执行该工具，检索结果并将其反馈给大型语言模型，以提供完整的答案。
5. **函数调用与工具调用：** 这两个术语在技术上是相同的。“函数调用”是 OpenAI 的术语；“工具调用”在 LangChain 和其他框架中使用得更广泛。

作者

IBM 技能网络团队