# 代理概述

想象一下，如果人工智能能像我们一样，具有推理能力，能够自主提出计划，批判性地评估这些想法，甚至将其付诸实践，那会是怎样的景象？就像电影《HER》中的人工智能萨曼莎，她不仅能与人进行深入的对话，还能够帮助西奥多规划日常生活，安排行程等。这样的设想一度只存在于科幻作品中，但现在通过代理技术，它似乎已经触手可及。

最近Yohei Nakajima的研究论文《Task-driven Autonomous Agent Utilizing GPT-4, Pinecone, and LangChain for Diverse Applications》展示了代理的突破创造性。在该研究中，作者提出了一个利用OpenAI的GPT-4语言模型、Pinecone向量搜索和LangChain框架的任务驱动的自主代理，该代理可以在多样化的领域中完成各种任务，生成基于完成结果的新任务，并在实时中优先处理任务。

那么Langchain中的代理是什么？为什么要使用代理？

#### 代理的定义

代理的核心思想是将语言模型作为推理引擎，依据其确定如何与外部世界交互以及应采取何种行动。这意味着，代理的行动序列是根据用户输入而变化的，无需遵循硬编码的顺序，例如，“先做A，再做B，然后做C”。相反，代理依据用户的输入和之前的行动结果来决定下一步采取何种行动。

#### 为什么使用代理?

代理和工具使用的概念紧密相关。工具能够将代理和外部数据源或计算（如搜索API、数据库）链接起来，这克服了语言模型的某些限制，例如语言模型无法直接理解你的数据，也不擅长处理数学运算。然而，工具的使用并不仅限于代理。你依然可以使用工具，将语言模型连接到搜索引擎，而无需代理。然而，使用代理的优势在于它们具有更高的灵活性、更强大的处理能力，能够更好地从错误中恢复，并处理复杂的任务。

例如，当与SQL数据库交互时，可能需要执行多个查询才能回答某些问题，而简单的工具序列可能很快就会遇到一些边缘情况。在这种情况下，代理这种更为灵活的框架能够更好地解决问题。

#### 代理的应用实现

在代理的典型实现中，用户首先提出一个请求，然后利用语言模型选择应使用的工具。接下来，执行对该工具的操作，获取观察结果，再将其反馈给语言模型，进行下一步操作，如此循环，直到满足停止条件。

停止条件有多种类型，最常见的是语言模型本身意识到任务已完成，应该给出回复。也可能有其他更具体的规则，比如，如果代理已经连续执行了五个步骤，但还没有得到最终答案，那么可能需要返回某些结果。在讨论代理的可靠性时，这些规则可以提供帮助。

代理的基本思想是选择一个工具，观察其输出，然后继续进行下一步。使用代理的方式，可以有效提高处理复杂任务的效率和准确性。

#### 代理的应用场景

Agent的一项重要功能是根据用户的问题来决定最佳的工具。它首先会检查输入，然后根据你初始化的工具来做出选择。接着，它会根据用户的问题，实际运行这些工具，从而生成预期的输出。通过这种方式，代理可以根据不同的用户输入，灵活地选择和使用不同的工具，从而提供丰富和多样的服务。

例如，假设您正在为一家电子商务企业构建一个聊天机器人。虽然您可以使用GBT4等语言模型进行聊天，但这些模型对于了解您的产品非常有限。为了解决这个问题，我们可以使用LangChain的向量存储功能，将产品数据存储在数据库中，并让语言模型可以访问这些数据。这样一来，聊天模型就可以更好地了解您的产品。

然而，单单了解产品还不够。如果聊天机器人在网页上运行，它还需要了解访问的上下文。这可能包括一些信息，比如访问者是新潜在客户还是现有客户，或者基于浏览历史来推荐产品。为了让语言模型在与客户交互时具备这些上下文信息，我们可以通过微服务向聊天模型提供这些信息。

通过让语言模型访问资源和上下文信息，语言模型可以与客户进行更好的交互，帮助企业转化客户并增加销售额。这些资源和信息可以通过代理的方式提供给语言模型。

#### 代理实现的方式：ReAct

一种主要且广泛应用的方法和策略被称为ReAct，这是“Reasoning and Acting（推理与行动）”的缩写。这一策略首次由普林斯顿大学在他们的一篇优秀论文中提出，现已被广泛应用于代理实现。

#### ReAct的优势

在许多应用场景中，ReAct策略已证明自己是非常有效的。例如，考虑这样一个问题：“除了Apple remote之外，还有哪些设备可以控制与Apple remote最初设计的互动的程序？”。最基本的提示策略是直接将这个问题交给语言模型处理，但ReAct策略赋予了代理更大的灵活性和实力。代理不仅可以使用语言模型，还可以连接到其他工具、数据源或计算环境，例如搜索API和数据库，以此来克服语言模型的某些局限性，例如对数据的不了解或数学运算能力有限。这样，即使遇到需要多次查询才能回答的问题，或者其他一些边界情况，代理也能够灵活应对，从而使其成为一种更强大的问题解决工具。

#### ReAct 如何工作?

ReAct策略的工作原理是什么呢？重申一下，代理的核心思想是将语言模型作为推理引擎。ReAct策略是将推理和行动结合在一起的方式。代理接收到用户的请求，然后使用语言模型选择要使用的工具。然后代理执行该工具的操作，观察结果，然后将这些结果反馈给语言模型。这个过程会持续进行，直到满足某些停止条件。停止条件可以有很多种，最常见的是语言模型认为任务已经完成，需要将结果返回给用户。这种方式使得代理具有更高的灵活性和强大的问题解决能力，这是ReAct策略的核心优势。

#### 实现代理应用的挑战

在实现代理应用的过程中，我们面临许多挑战，以下列举了几个主要的：

首先，使代理在适当的场景下使用工具是我们面临的一个基本挑战。如何在合适的情况下让代理采用恰当的工具，并优化其使用效果呢？在ReAct论文中，通过引入推理的角度，以及使用”CoT 思考链”的提示方式，我们寻求解决这个问题。在实际操作中，我们常常需要明确告知代理可使用的工具，以及通过这些工具能克服的限制。所以，工具的描述信息也非常重要，如果我们希望代理能用特定的工具，就需要提供足够的上下文信息，使代理能理解工具的优点和应用场景。

其次，对于工具的选择，我们需要进行检索。这一步骤可以解决上述的问题。我们可以运行一些检索步骤，例如嵌入式搜索查找，以获取可能的工具，然后将这些工具传递给提示，由语言模型进行后续步骤。

此外，提供相关的示例也是一种有效的方法。选择与当前任务类似的示例，通常比随机示例更有帮助。相同地，检索最相关的示例也有巨大的潜力。

最后，我们还需要注意避免在不需要的情况下使用工具。可以在提示中加入相关信息或提醒，告诉代理在对话时不必使用工具。

#### 实现代理应用的实用技巧

在解决这些挑战的过程中，我们总结出了一些实用的技巧：

首先，结构化的响应更易于解析。通常情况下，你提问的响应越结构化，解析起来就越容易。语言模型在编写JSON方面表现得很好，因此我们将一些代理转换为使用JSON格式。

其次，我们引入了输出解析器的概念。输出解析器封装了解析响应所需的全部逻辑，并以尽可能模块化的方式实现。另一个相关的概念是，输出解析器可以重试和修复错误。如果有格式错误的模式，你可以通过将输出和错误传递给它来显式地修复响应。

此外，记住之前的步骤也是很重要的。最基本的方法是在内存中保留这些步骤的列表。然而，在处理长时间运行的任务时，会遇到一些上下文窗口的问题。我们已经找到了一种解决方法，即使用一些检索方法来获取之前的步骤，并将其放入上下文中。

最后，在处理API时，我们经常遇到观察结果太长的问题。因为API通常会返回非常大且难以放入上下文的JSON数据。常见的解决方法是对其进行解析，可以简单地将该大数据块转换为字符串，并将前1000个字符作为响应。

#### 最新的代理应用项目及其对改进的探索

近期的代理应用项目研发涉猎广泛，主要集中在如何改善代理的各种工作方式上。下面介绍四个具有代表性的项目。

1. AutoGPT：AutoGPT 的目标设置有别于 ReAct 代理的重大不同。AutoGPT 的追求在于如何增加 Twitter 的关注者数量或实现其他类似的开放性、广泛性和长期性目标。相较之下，ReAct 代理则专注于实现短期内可量化的目标。为了实现这样的目标，AutoGPT引入了长期记忆的概念，促进代理与工具之间的互动，这有助于提升代理的规划和执行效率。
2. Baby AGI：Baby AGI 的研发采用了逐步解决子问题的方法，以提升代理的规划和执行能力。这一项目明确了策划和执行步骤的定义，这一创新为提升长期目标代理的可行性和关注度提供了有益的思考途径。最初，Baby AGI 的策略实现主要依靠自主设定，然而现在已经开始融入了各种工具，从而优化代理执行计划的能力。
3. Camel：Camel 项目的一项主要创新是在模拟环境中进行代理之间的交互。通过这种方法，可以对代理进行评估和测试，并且可以作为一种娱乐手段。这种方法为检测代理交互提供了一种无需人工干预的方式，能够有效地测试代理模型。
4. Generative Agents：该项目的目标是通过构建一个复杂的模拟环境，让 25 个不同的代理在这个环境中进行互动，从而实现生成型代理。项目同时也注重处理代理的记忆和反思能力，代理能够通过记忆中的事件来指导下一步的行动，并在反思环节对最近的事件进行评估和更新。这种基于反思的状态更新机制适用于各种类型的记忆，例如实体记忆和知识图谱，从而提高代理对环境的建模能力。

#### 最简单的代理示例

首先，让我们加载语言模型。

from langchain.agents import load\_tools  
from langchain.agents import initialize\_agent  
from langchain.agents import AgentType  
from langchain.llms import OpenAI

llm = OpenAI(temperature=0)

接下来，让我们加载一些要使用的工具。请注意，llm-math工具使用了一个LLM，所以我们需要传递进去。

tools = load\_tools(["serpapi", "llm-math"], llm=llm)

最后，让我们用这些工具、语言模型和我们想要使用的代理类型来初始化一个代理。

agent = initialize\_agent(tools, llm, agent=AgentType.ZERO\_SHOT\_REACT\_DESCRIPTION, verbose=True)

现在，让我们来测试一下吧！

agent.run("Who is Leo DiCaprio's girlfriend? What is her current age raised to the 0.43 power?")

> Entering new AgentExecutor chain...  
 I need to find out who Leo DiCaprio's girlfriend is and then calculate her age raised to the 0.43 power.  
 Action: Search  
 Action Input: "Leo DiCaprio girlfriend"  
 Observation: Camila Morrone  
 Thought: I need to find out Camila Morrone's age  
 Action: Search  
 Action Input: "Camila Morrone age"  
 Observation: 25 years  
 Thought: I need to calculate 25 raised to the 0.43 power  
 Action: Calculator  
 Action Input: 25^0.43  
 Observation: Answer: 3.991298452658078  
   
 Thought: I now know the final answer  
 Final Answer: Camila Morrone is Leo DiCaprio's girlfriend and her current age raised to the 0.43 power is 3.991298452658078.  
   
 > Finished chain.  
  
  
  
  
  
 "Camila Morrone is Leo DiCaprio's girlfriend and her current age raised to the 0.43 power is 3.991298452658078."