第 1 章

LangChain 与大型语言模型

LangChain 是一个用于开发基于语言模型的应用程序的开发框架，它旨在帮助开发者使用大型语言模型(LLM)构建端到端的应用程序，并提供了一系列工具、组件和接口来实现这一目标。本章将详细讲解 LangChain 与大型语言模型的基础知识。

1.1 LangChain 与大型语言模型


扫码看视频

LangChain 是一个开源框架，开发者可通过它将大型语言模型(Large Language Model，LLM)与其他数据源和计算能力结合起来，以创建能够理解和生成自然语言的应用程序。LangChain 的设计目标是简化开发过程，通过提供模块化的抽象和针对特定用例的链，让开发者可以轻松地开始特定项目的开发，并支持这些链的个性化定制。

1.1.1 LangChain 的基本概念与原理

LangChain 是一款用于开发由语言模型驱动的应用程序的框架，它旨在简化大型语言模型(LLM)的集成与应用。这个框架提供了工具和接口，使开发者能够轻松地将大型语言模型集成到各种应用程序中，进而利用这些模型强大的语言理解和生成能力。

LangChain 的核心优势在于其模型接口的统一性。它封装了多种大型语言模型的 API 接口，允许开发者无缝切换不同的模型而无须重写大量代码。此外，LangChain 还对提示管理、内存管理以及信息检索索引等关键功能进行了优化，以提升应用程序的性能和用户体验。

LangChain 的设计目标是实现数据感知和主动性，允许语言模型与其环境进行交互。它提供了组件化的抽象，为处理语言模型所需的组件(如提示模板、信息检索索引等)提供了易于使用的实现。LangChain 还提供了用例特定链的概念，这些链通过将组件以特定方式组合，以便有效地解决特定用例的需求。

1.1.2 大型语言模型介绍

大型语言模型是自然语言处理(Natural Language Processing，NLP)领域的一项重要技术。这些模型通常由深度学习算法驱动，能够处理和生成自然语言文本。大型语言模型的主要特点如下。

- 预训练(Pre-training)：大型语言模型通常在大量的文本数据上进行预训练，这些数据包括书籍、文章、网页等。预训练过程使模型能够学习语言的模式和结构。
- 参数数量(Parameter Count)：大型语言模型拥有大量的参数，这些参数可以是数十亿甚至数千亿。参数数量是模型能力的一个重要指标，更多的参数通常意味着模型能够捕捉并理解更复杂的语言特征。

- 深度学习架构(Deep Learning Architectures)：这些模型通常基于深度学习架构，如变换器(Transformer)，它们使用注意力机制来处理序列数据。
- 微调(Fine-tuning)：在预训练之后，大型语言模型可以在特定任务上进行微调，以提高在特定领域的性能。微调过程涉及在特定任务的数据集上进一步训练模型。
- 多任务学习(Multi-task Learning)：一些大型语言模型能执行多种 NLP 任务，如文本分类、情感分析、机器翻译、问答等，而无须针对每个任务单独训练。
- 生成能力(Generative Abilities)：大型语言模型不仅能理解输入的文本，还能够生成新的文本。这使它们在创意写作、对话系统等领域具有广泛的应用。
- 上下文理解(Contextual Understanding)：这些模型能够理解上下文中的信息，从而生成与上下文相关的响应或文本。
- 可扩展性(Scalability)：随着计算资源的增加，大型语言模型可以变得更大、更复杂，从而提高性能。
- 商业和研究应用(Commercial and Research Applications)：大型语言模型被广泛应用于商业产品和研究项目中，包括搜索引擎、推荐系统、虚拟助手以及教育工具等多个领域。

随着人工智能技术的不断发展，大型语言模型正在不断进化，它们在理解和生成自然语言方面的能力也在不断提高，为人工智能领域带来了新的可能性。

1.1.3 LangChain 在大型语言模型应用中的作用

在大型语言模型应用中，LangChain 的主要作用如下。

- 集成外部数据源：LangChain 允许大型语言模型访问未包含在训练数据集中的特定外部数据源，如内部文档、电子邮件或数据集，这些外部文档在 LangChain 中统称为“索引”。
- 文档加载器：LangChain 提供多种文档加载器，可以轻松地从文件存储服务(如 Dropbox、Google Drive 和 Microsoft OneDrive)、网络内容、协作工具、数据库等来源导入数据。
- 矢量数据库：LangChain 能利用矢量数据库技术，将数据点转换为矢量嵌入，提高查询效率，并且可以存储每个矢量的元数据，增强搜索能力。
- 文本分割器：LangChain 能将大型文本文档分割为具有语义意义的较小部分，以减少计算需求并提高处理速度。
- 检索增强生成 (RAG)：LangChain 提供了检索器模块，接收字符串查询作为输入，

并返回文档列表作为输出，增强了模型的检索与生成能力。

- 代理：LangChain 代理可以将给定的语言模型作为“推理引擎”，确定要采取哪些行动，并构建链以执行任务。
- 工具集成：LangChain 提供了工具集成功能，允许代理与现实世界的信息交互，以扩展或改进可以提供的服务。
- 标准化模型接口：LangChain 提供标准化模型接口，允许开发者自由切换不同的模型，包括文本生成模型和文本嵌入模型。
- 提示词模板：LangChain 提供帮助开发者构建提示词的工具，使模型返回期望的内容。
- 索引和检索：LangChain 提供索引工具，使文档结构化，便于 LLM 直接与文档交互，其中的检索器用于检索文档数据。

通过这些功能，LangChain 为大型语言模型构建了一个强大的框架，使开发者可以构建功能更加强大的应用程序，如智能客服、个性化推荐、知识图谱构建等。LangChain 通过简化开发流程，并提供模块化的工具和接口，极大地推动了大型语言模型在各种应用场景下的实用效能与运行效率。

1.2 LangChain 入门指南


扫码看视频

LangChain 是一个开源框架，从事人工智能的开发者可通过它将如 GPT-4 这样的大语言模型与外部计算和数据来源结合起来，使应用程序具备以下特性。

- 上下文感知：将语言模型连接到上下文来源(包括提示指令、少量示例、用于建立响应基础的内容等)。
- 推理：依赖语言模型进行推理(根据提供的上下文进行回答的方式、采取的行动等)。

1.2.1 安装 LangChain

LangChain 能够构建基于语言模型驱动，具备上下文感知和推理能力的应用程序。安装 LangChain 后，用户可通过 pip 或 conda 命令，利用 langchain-community、langchain-core、langchain- experimental 等附加包，以及 LangServe、LangChain CLI 和 LangSmith SDK 等辅助工具进行更深层次的集成、部署和开发。LangChain 的价值在于它简化了应用程序的开发、生产和部署流程，并提供了丰富的附加模块和工具。

要安装 LangChain，可以使用以下方法之一实现。

(1) 通过 pip 安装，安装命令如下：

```
pip install langchain
```

这是安装 LangChain 的基本要求。但是，如果需要与各种模型提供程序、数据存储等进行集成，则需安装相应的依赖项，此时，LangChain 的真正价值才能体现。用户需根据需求单独安装这些特定集成的依赖项。

(2) 通过 conda 安装的命令如下：

```
conda install -c conda-forge langchain
```

(3) 从源代码安装。如果想要从源代码安装，可以克隆存储库，并确保在目录 PATH/TO/REPO/ langchain/libs/langchain 下运行。安装命令如下：

```
pip install -e .
```

(4) 安装 LangChain community 包。LangChain community 包包含第三方集成，它会在安装 LangChain 时自动安装，也可单独使用。安装命令如下：

```
pip install langchain-community
```

(5) 安装 LangChain core 包。LangChain core 包包含 LangChain 生态系统中其余部分使用的基本抽象和 LangChain Expression Language，安装 LangChain 时会自动安装，也可单独安装。安装命令如下：

```
pip install langchain-core
```

(6) 安装 LangChain experimental 包。LangChain experimental 包包含用于研究和实验目的的实验性 LangChain 代码，安装命令如下：

```
pip install langchain-experimental
```

(7) 安装 LangServe。LangServe 帮助开发人员将 LangChain 可运行程序和链部署为 REST API。可使用以下命令进行安装：

```
pip install "langserve[all]"
```

或者分别安装客户端和服务器依赖项，安装命令如下。

```
pip install "langserve[client]"
pip install "langserve[server]"
```

(8) 安装 LangChain CLI。LangChain CLI 对使用 LangChain 模板和其他 LangServe 项目非常有用，可使用以下命令安装：

```
pip install langchain-cli
```

(9) 安装 LangSmith SDK。LangSmith SDK 会在安装 LangChain 时自动安装。如果未自动安装用 LangSmith，可使用以下命令安装：

```
pip install langsmith
```

在实际应用中，应选择适合自己的安装方式，并根据需要安装附加包和工具。

1.2.2 LangChain 框架的组成

LangChain 框架专为开发语言模型驱动的应用程序而设计，包含以下几个主要部分。

- LangChain Libraries(LangChain 库)：提供 Python 库和 JavaScript 库，包含用于各种组件的接口与集成，以及用于将这些组件组合成链和代理的基本框架与工具，还有针对常见任务的预制链实现模板。
- LangChain Templates(LangChain 模板)：提供一组易于部署的参考架构，适用于各类任务。
- LangServe：一个专门用于将 LangChain 链部署为 REST API 的库。
- LangSmith：一个开发平台，支持在任何大型语言模型框架上调试、测试、评估及监控构建在其上的链且与 LangChain 实现无缝集成。

上述组件共同简化了整个应用程序生命周期，具体说明如下。

- 开发(Develop)：在 LangChain/LangChain.js 中编写自己的应用程序，可将模板作为参考以快速入门。
- 生产(Productionize)：使用 LangSmith 检查、测试和监视链，以便能够不断改进并放心地进行部署。
- 部署(Deploy)：使用 LangServe 将任何链转换为 API。

LangChain Libraries 本身由几个不同的软件包组成，具体说明如下。

- langchain-core(LangChain 核心)：包含构建 LangChain 应用的基础抽象类、接口，以及用于定义组件组合逻辑的核心语法与表达规则。
- langchain-community(LangChain 社区库)：包含第三方集成。
- langchain(LangChain)：包含构成应用程序认知架构的链、代理和检索策略等组件。

注意

LangChain 不是 OpenAI 的产品，而是由 LangChain Inc.开发的框架，用于构建由语言模型驱动的应用程序。OpenAI 和 LangChain Inc.是两个不同的组织，前者专注于人工智能模型的研究与开发，后者专注于语言模型应用框架的构建。

第2章

第一个 LangChain 程序

在深入探讨了 LangChain 的基础概念后，现在让我们开始来构建“第一个 LangChain 程序”。这将是一次实践 LangChain 强大功能的机会。通过将理论应用到实践中，我们将一步步地实现一个完整的应用程序。该程序不仅会展示 LangChain 的核心优势，还将揭示如何将大型语言模型的潜力转化为实际的解决方案。本章将详细讲解 LangChain 的开发流程，并介绍如何开发第一个大型 LangChain 程序所需的知识。

2.1 LangChain 开发流程介绍

扫码看视频

LangChain 的主要功能是构建和部署基于大型语言模型的应用程序。其通用的开发流程如下。

(1) 需求分析：明确应用程序的目标和需求，包括预期功能、用户交互方式以及性能标准。

(2) 环境设置：安装 LangChain 及其必要的依赖项，配置开发环境，如 Jupyter Notebook 或其他 IDE。

(3) 选择语言模型：根据应用程序的需求，选择一个合适的大型语言模型。LangChain 支持多种模型，如 GPT-3、BERT 等。

(4) 定义数据源：确定应用程序需要访问的数据源。LangChain 能够集成多种数据源，如文件、数据库、APIs 等。

(5) 构建索引：使用 LangChain 的索引构建工具，将数据源中的数据转换为易于检索的格式。

(6) 设计提示模板：创建提示模板，这些提示模板将指导语言模型生成特定的输出。

(7) 构建代理：设计代理(Agents)，这些代理能够执行特定任务，如数据检索、信息提取等。

(8) 创建 LangChain 链：使用 LangChain 的链(Chains)功能，将不同的组件(如索引、代理、提示模板等)组合起来，构建完整的应用程序逻辑。

(9) 开发交互界面：开发用户界面，允许用户与应用程序交互，可以是命令行界面、Web 界面或移动应用。

(10) 测试：对应用程序进行测试，确保所有功能按预期工作，并修复发现的问题。

(11) 优化：根据测试结果和性能指标，对应用程序进行优化，提高效率并优化用户体验。

(12) 部署：将应用程序部署到生产环境，确保系统的稳定性和可扩展性。

(13) 监控和维护：监控应用程序的性能，并定期进行更新和维护，以适应新的需求和解决潜在问题。

LangChain 的开发流程强调模块化和灵活性，使开发者能够根据特定需求快速构建和定制应用程序。通过 LangChain，开发者可以利用大型语言模型的强大能力，创建智能、高效的应用程序。

2.2 使用 LangChain 构建应用程序

使用 LangChain，开发者能够构建与外部数据源和计算过程无缝集成的应用程序。在接下来的内容中，我们将演示如何使用不同的方法实现这种集成。

扫码看视频

2.2.1 LLM 链

LLM 链是将大语言模型与其他组件按照一定的顺序和规则连接在一起的结构。在 LangChain 中，这些组件包括输入提示模板(如 ChatPromptTemplate)、语言模型(如 ChatOpenAI)、输出解析器(如 StrOutputParser)等。LLM 链的目的是将多个处理步骤有序地连接在一起，形成一个完整的处理流程。用户可以通过这个链提供输入，链中的组件逐步处理输入，最终生成输出。这种组件化的设计使用户能够更灵活地构建自己的自然语言处理应用，并能根据需要定制不同的处理步骤和模型。

在 LangChain 中，LLM 链通常用于构建可以连接外部数据源和计算资源到语言模型的应用程序。链的每个步骤都可以执行不同的任务，如处理提示、调用语言模型、解析输出等。LangChain 官网提供了两种模型作为示例：OpenAI(通过 API 提供的流行模型)和本地的开源模型 Ollama。在本书中，只展示如何使用 OpenAI 模型。

(1) 首先，通过以下命令安装 LangChain x OpenAI 集成包：

```
pip install langchain-openai
```

(2) 使用 OpenAI API 时需要提供 API 密钥，这是用于验证身份并访问 OpenAI 服务的凭据。获取 OpenAI 密钥后，通过以下命令将其设置为环境变量：

```
export OPENAI_API_KEY="..."
```

(3) 初始化模型：

```
from langchain_openai import ChatOpenAI
llm = ChatOpenAI()
```

如果不想设置环境变量，可以在初始化 OpenAI LLM 类时直接通过 openai_api_key 参数传递密钥：

```
from langchain_openai import ChatOpenAI
llm = ChatOpenAI(openai_api_key="...")
```

(4) 安装并初始化所选的大语言模型后，可以尝试使用它，例如，询问它“LangSmith

是什么”。由于这是训练数据中不存在的内容，它可能无法提供很好的响应：

```
llm.invoke("how can langsmith help with testing?")
```

(5) 还可以使用提示模板引导响应，提示模板用于将原始用户输入转换为更适合 LLM 的输入，命令如下：

```
from langchain_core.prompts import ChatPromptTemplate
prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a world-class technical documentation writer."),
    ("user", "{input}")
])
```

(6) 现在，将前面提到的组件(包括 ChatPromptTemplate、ChatOpenAI 和 StrOutputParser)组合成一个简单的 LLM 链，以构建一个完整的处理流程，命令如下：

```
chain = prompt | llm
```

(7) 现在可以调用该链并提出同样的问题。尽管它可能不知道答案，但能以技术撰写人更正式的语气回应，命令如下：

```
chain.invoke({"input": "how can langsmith help with testing?"})
```

在 LangChain 中，ChatModel 是一种模型，其输出是一条消息(ChatMessage)。然而，在某些情况下，为了方便处理，用户可能更倾向于将输出转换为字符串形式。为了实现这一点，可以添加一个简单的输出解析器，命令如下：

```
from langchain_core.output_parsers import StrOutputParser
output_parser = StrOutputParser()
```

注意

输出解析器是一种组件，用于将一个输出类型转换为另一种输出类型，通常是更易处理的类型。在这里，将输出消息(ChatMessage)转换为字符串(String)形式。它可以通过在 LangChain 流程中添加一个名为“StrOutputParser”的输出解析器来完成。

这个输出解析器的作用是将聊天消息转换为字符串，使后续处理更加方便，因为字符串是一种常见的文本表示形式，易于处理和输出，它是为了满足用户在处理输出时的特定需求。然后将其添加到先前的链中，命令如下：

```
chain = prompt | llm | output_parser
```

现在调用它并提出同样的问题，此时，答案将是一个字符串(而不是一个 ChatMessage)，命令如下：

```
chain.invoke({"input": "how can langsmith help with testing?"})
```

我们已经成功设置了一个基本的 LLM 链，其中涉及提示、模型和输出解析器的基础知识。

2.2.2 检索链

检索链(Retrieval Chain)是 LangChain 中的一个概念，专门用于处理对话系统中的信息检索任务。在对话系统中，当用户提出问题时，系统可能需要从大量的文档或数据中检索相关信息，然后将这些信息传递给大型语言模型进行进一步处理。

检索链的工作流程如下。

(1) Retriever(检索器)：检索链的第一步是通过检索器获取相关文档或信息。检索器可以基于多种方式工作，如使用向量存储、数据库查询等。在 LangChain 中，用户可以配置一个检索器，使其能够从向量存储中检索相关文档。

(2) Documents(文档)：检索到的文档被传递到下一步。这些文档包含对用户问题有用的信息。

(3) LLM 处理：下一步是将检索到的文档和用户的问题传递给大型语言模型。这样，LLM 可以基于上下文信息提供更准确的答案。

(4) 生成答案：最终，大型语言模型处理输入并生成对用户问题的答案，再将这个答案返回给用户。

检索链的功能在处理大量信息时显得尤为重要，因为它允许系统首先从整个数据集中检索最相关的部分，然后仅将这些部分传递给大型语言模型。这提高了系统的性能和响应速度，尤其是在处理大规模数据集时。

为了正确回答原始问题(“how can langsmith help with testing?”)，我们需要向 LLM 提供额外的上下文信息，这可以通过检索(Retrieval)来实现。检索链在有大量数据需要传递给 LLM 时非常有用，我们可以使用检索链提取最相关的信息并将其传递给 LLM。在这个过程中，将从一个检索器中查找相关文档，然后将它们传递到提示中。检索器可以由任何东西支持，如 SQL 表、互联网等，但在这个实例中，我们将填充一个向量存储并将其用作检索器。

(1) 使用 WebBaseLoader 加载要进行索引的数据，这需要安装 BeautifulSoup：

```
pip install beautifulsoup4
```

(2) 导入并使用 WebBaseLoader。

```
from langchain_community.document_loaders import WebBaseLoader
loader = WebBaseLoader("https://docs.smith.langchain.com/overview")
```