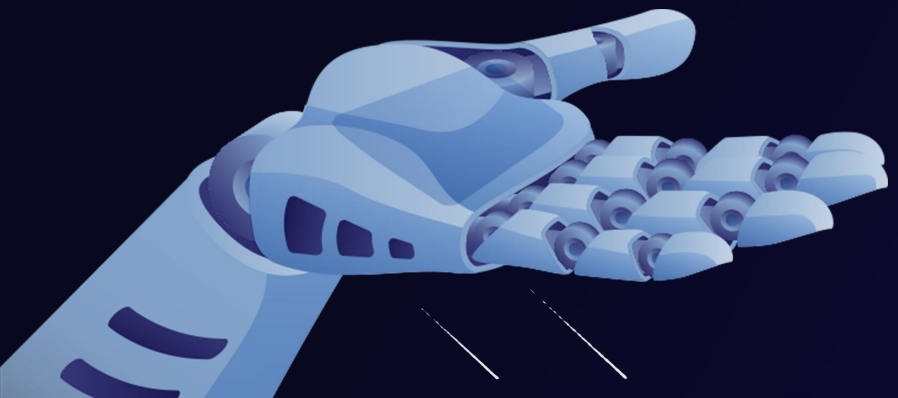
第二章：LangChain工具及应用



CONTENTS

2.1 LangChain简介

2.2 模型、提示和解析器

2.3 记忆机制

2.4 模型链结构

2.5 代理机制

2.6 RAG: 访问本地文档

2.7 搭建基于本地知识库的问答系统

2.1 LangChain简介

why：尽管大语言模型（LLM）具备强大的能力，大幅简化了应用程序的开发，但要基于大模型构建一个完整的应用程序仍需要较大的代码量。

what：为解决这一问题，LangChain 应运而生。LangChain 是专为大模型开发设计的开源框架，致力于充分利用大模型的各种强大功能。它提供了模型、提示、记忆、链、代理等功能，使开发者能够轻松地构建应用程序，最大化地发挥大模型的潜力。目前，掌握 LangChain 已成为大模型开发者的必备技能之一。

模型(Models)：集成各种语言模型与向量模型。

链(Chain)：将模块组合的能力。

提示(Prompts)：管理发送到模型的指令。

代理(Agent)：调用工具的能力。

解析(Parsers)：解析模型返回的结果。

检索增强(RAG)：基于本地文档问答的能力。

记忆(Momory)：管理大模型的记忆能力。

2.2 模型、提示和解析器

提示

实现对提示（Prompt）的封装，使用方式如下：

```
from langchain.prompts import
ChatPromptTemplate

# step1：构造一个提示模版字符串
prompt_template_str = """请把下面三个反引号
分隔的文本转换成{style}的风格。\
文本: ```{text}```
"""

# step2：构造提示模板
prompt_template =
ChatPromptTemplate.from_template(prompt_tem
plate_str)

# step3：使用提示模板
user_comment = """
上次我在你们店铺买了一双鞋子，结果到货后
发现一只白的，一直黑的，简直不敢相信，太
让人生气了。我现在要求退货。
"""
style = """
平静、尊敬、有礼貌
"""
user_messages =
prompt_template.format_messages(style=style,
text=user_comment)
```

模型

实现对大模型的封装，调用方式如下：

```
from langchain.chat_models import
ChatOpenAI

chat =
ChatOpenAI(model_name=model_name,
temperature=0.0,
openai_api_key=openai.api_key)
response = chat(user_messages)
```

除了ChatOpenAI，还有BedrockChat, AzureChatOpenAI, ChatGooglePalm, ChatMlflow等。

解析

支持构造合适的提示，并对结果解析到特定格式，以下是一个对输出解析成字典的例子：

step1：编写提示。

step2：依据输出结构构造标准化输出模式类（ResponseSchema）与解析类（StructuredOutputParser）。

step3：依据StructuredOutputParser生成完整的提示。

step4：完整的提示送入大模型返回结果。

step5：依据StructuredOutputParser对输出的结果进行解析。

2.3 记忆机制

大模型训练完成之后，掌握的知识（参数）是固定的，可以理解为**长期记忆**。
用户每次和大模型的聊天内容，大模型并不会记住，为了让大模型记住这部分内容，需要人为地将聊天内容存储，并在下次对话时将前期的交互内容一起输入给大模型，这样大模型好像有了记忆，这个记忆也称为**短期记忆**。
langchain提供了短期记忆的多种存储机制，包括对话缓存储存、对话缓存窗口储存、对话令牌缓存储存、对话摘要缓存储存等。

对话缓存储存	对话缓存窗口储存	对话令牌缓存储存	对话摘要缓存储存
对话缓存储存的python类是ConversationBufferMemory，他可以完整的将对话历史保存下来，缺点是对话历史如果过长会增加大模型的成本。	对话缓存窗口储存的python类是ConversationBufferWindowMemory(k=1)，他通过参数k控制最近保留的对话数量，对中文支持的不够好。	对话令牌缓存储存的python类是ConversationTokenBufferMemory(llm=llm, max_token_limit=50)，他通过参数llm将对话转换为token，通过max_token_limit控制最近的token数量。	对话摘要缓存储存的python类是ConversationSummaryBufferMemory(llm=llm, max_token_limit=70)，他通过max_token_limit控制最近的原文输入数量，原文部分之前的对话将通过参数llm转换为摘要输入。

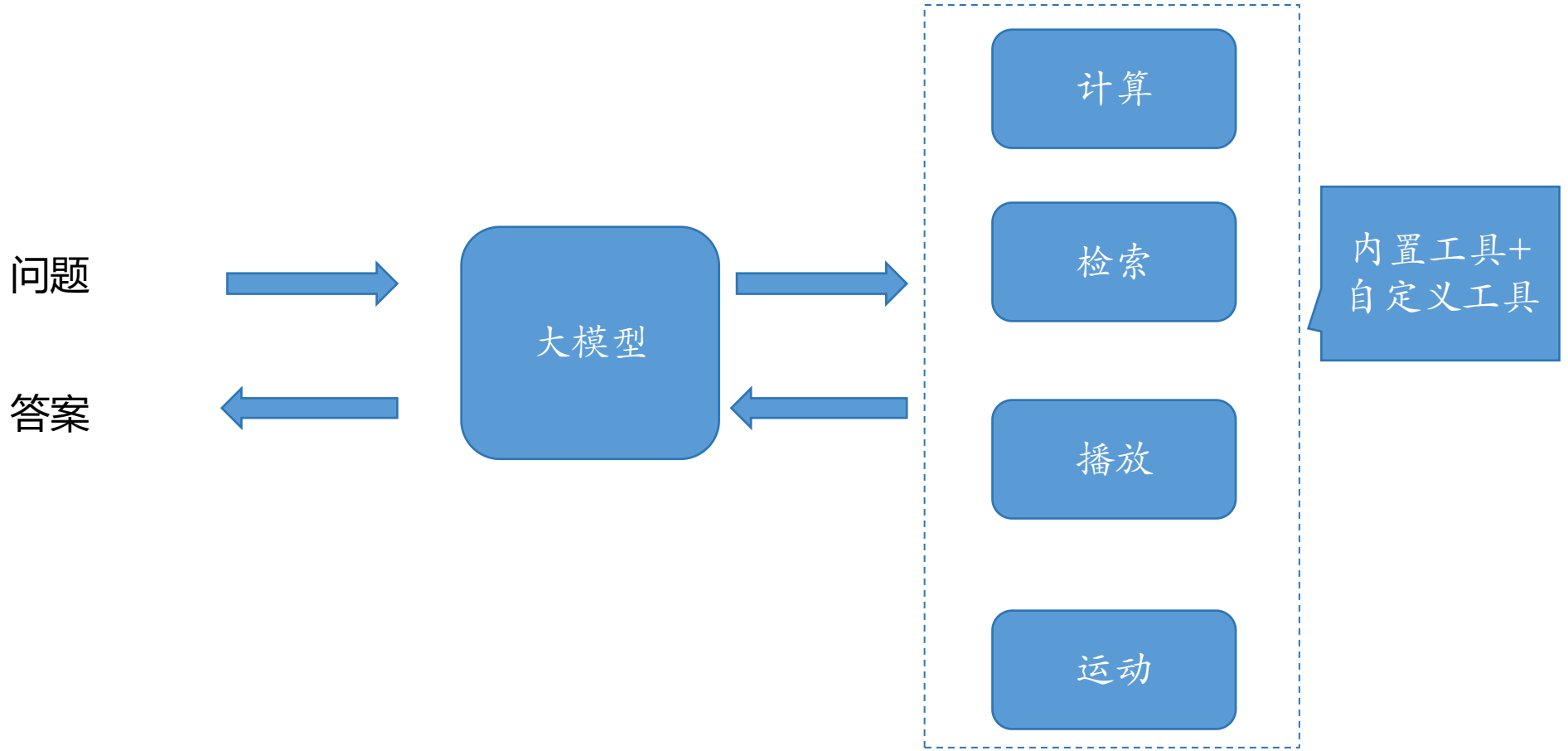
2.4 模型链结构

提示、大模型、工具函数是大模型应用程开发最常见的组件，langchan提供了链的能力，可以方便的将它们组合起来。常见的链有大语言模型链、简单顺序链、顺序链和路由链。

大语言模型链	简单顺序链	顺序链	路由链
基于LLMChain 将提示模板和大模型进行组合。接收输入并拼接prompt，然后调用大模型回答，属于基本链。	基于SimpleSequentialChain 将基本链进行顺序拼接。特点是每个基本链只有一个输入和一个输出，一个步骤的输出是下一个步骤的输入。	基于SequentialChain 将基本链进行顺序拼接。可以完成相对复杂的组合，允许每个基本链有多个输入和多个输出。	基于MultiPromptChain、LLMRouterChain和RouterOutputParser 将基本链进行路由调配。根据用户输入的不同自动导航至合适的链上。

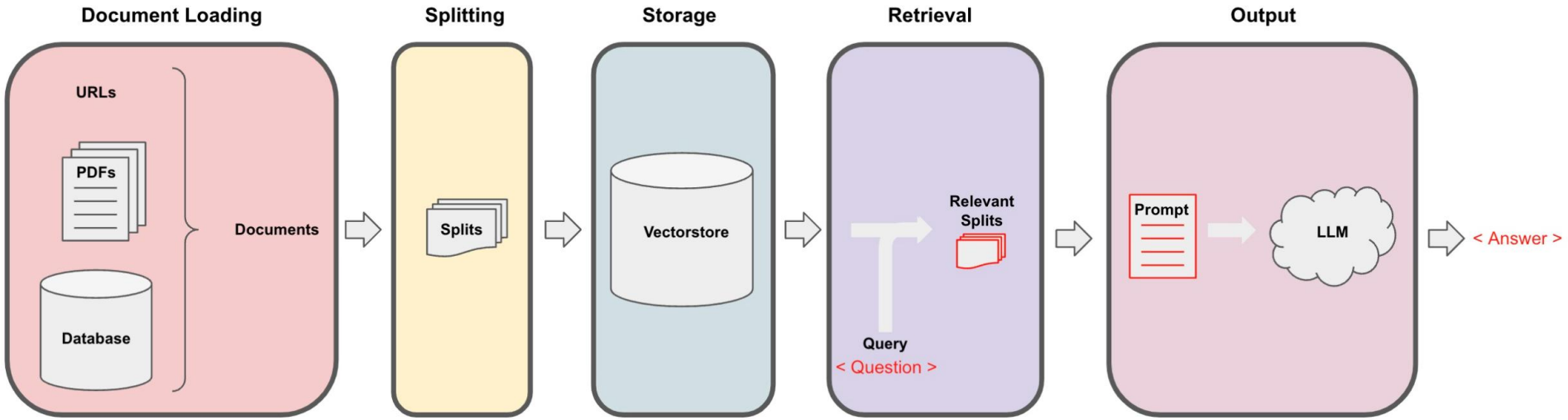
2.5 代理机制

大模型虽然在语义理解和生成方面非常强大，但是缺乏检索、计算等传统程序可以轻松处理的问题。代理机制是指通过大模型调用外部工具或函数，比如检索、计算、播放甚至运动（机器人）等等。如果将大模型比作大脑的话，代理可以比作四肢，通过代理机制可以大大扩展大模型的能力边界。



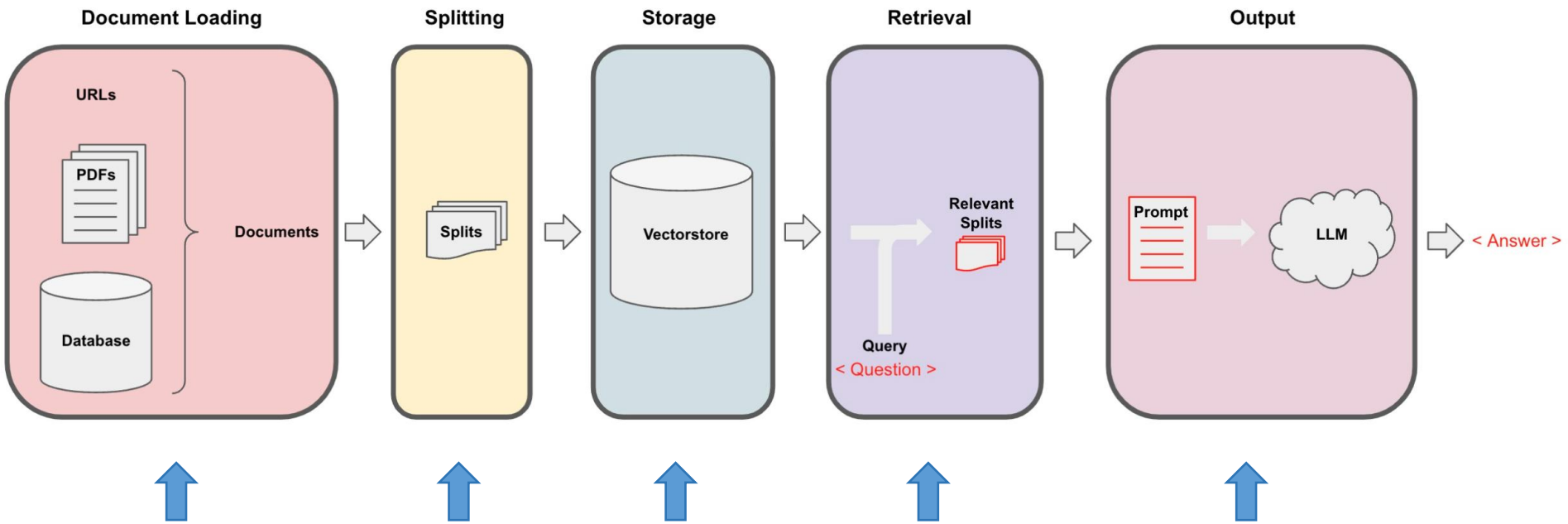
2.6 RAG: 访问本地文档

大模型虽然在预训练阶段学习到了互联网海量的知识，但是对于个人数据却无法回答。如果想要大模型回答个人数据相关问题可以使用检索增强技术RAG。简单的说，RAG技术首先拿用户的问题去个人数据库匹配相关的文档片段，然后送入大模型进行回答。具体来说，RAG技术分为几个环节，包括文档加载、文档分隔、向量存储、文档检索、大模型回答。



pdf、youtube音频、网页、markdown等多种加载方式	基于字符分隔、基于token分隔、分隔markdown	openai、google等多种向量化方法	基本语义相似度、最大边际相关性、过滤元数据、LLM辅助检索	Stuff MapReduce、 Refine、 MapRerank 四种问答策略

2.7 搭建基于本地知识库的问答系统



加载： 将原始信息加工称markdown格式数据

分隔： 按标题和段落分隔

存储： openai向量存储

检索： 根据相似性进行检索

LLM： 一次性将全部文档送入大模型

感谢您的观看