



創新研發部與
聯合實驗室



BoardWare

大语言模型应用 介绍与分享



以 ChatGPT 为代表的大语言模型在问题回答、文稿撰写、代码生成、数学解题等任务上展现出了强大的能力，引发了研究人员广泛思考如何利用这些模型开发各种类型的应用，**并修正它们在推理能力、获取外部知识、使用工具及执行复杂任务等方面的不足。**

此外，研究人员还致力于如何将文本、图像、视频、音频等多种信息结合起来，**实现多模态大模型**，这也成了一个热门研究领域。鉴于大语言模型的参数量庞大，以及针对每个输入的计算时间较长，优化模型在推理阶段的执行速度和用户响应时长也变得至关重要。

目录

Contents

1.1

推理规划

1.2

综合应用框架

目录

Contents

1.1

推理规划

1.2

综合应用框架



随着语言模型规模的不断扩大，其也具备了丰富的知识和强大的语境学习能力。然而，仅仅通过扩大语言模型的规模，**并不能显著提升推理（Reasoning）能力**，如常识推理、逻辑推理、数学推理等。**通过示例（Demonstrations）**或者明确指导模型在面对问题时如何逐步思考，促使模型在得出最终答案之前生成中间的推理步骤，可以显著提升其在推理任务上的表现。这种方法被称为思维链提示**（Chain-of-Thought Prompting）[1]**。

同样地，面对复杂任务或问题时，大语言模型可以展现出良好的**规划（Planning）能力**。通过引导模型首先将复杂的问题分解为多个较为简单的子问题，然后逐一解决这些子问题，可使模型得出最终解答，这种策略被称为**由少至多提示（Least-to-Most Prompting）[2]**。思维链提示和由少至多提示这两种方式，可以提升大语言模型的推理规划能力。



思维链提示 (**Chain-of-Thought Prompting**) 方式如图1.1 所示, 标准少样本提示 (Standard Few-shot Prompting) 技术在给模型的输入里面提供了 k 个[问题, 答案] 对, 以及当前问题, 由模型输出答案。

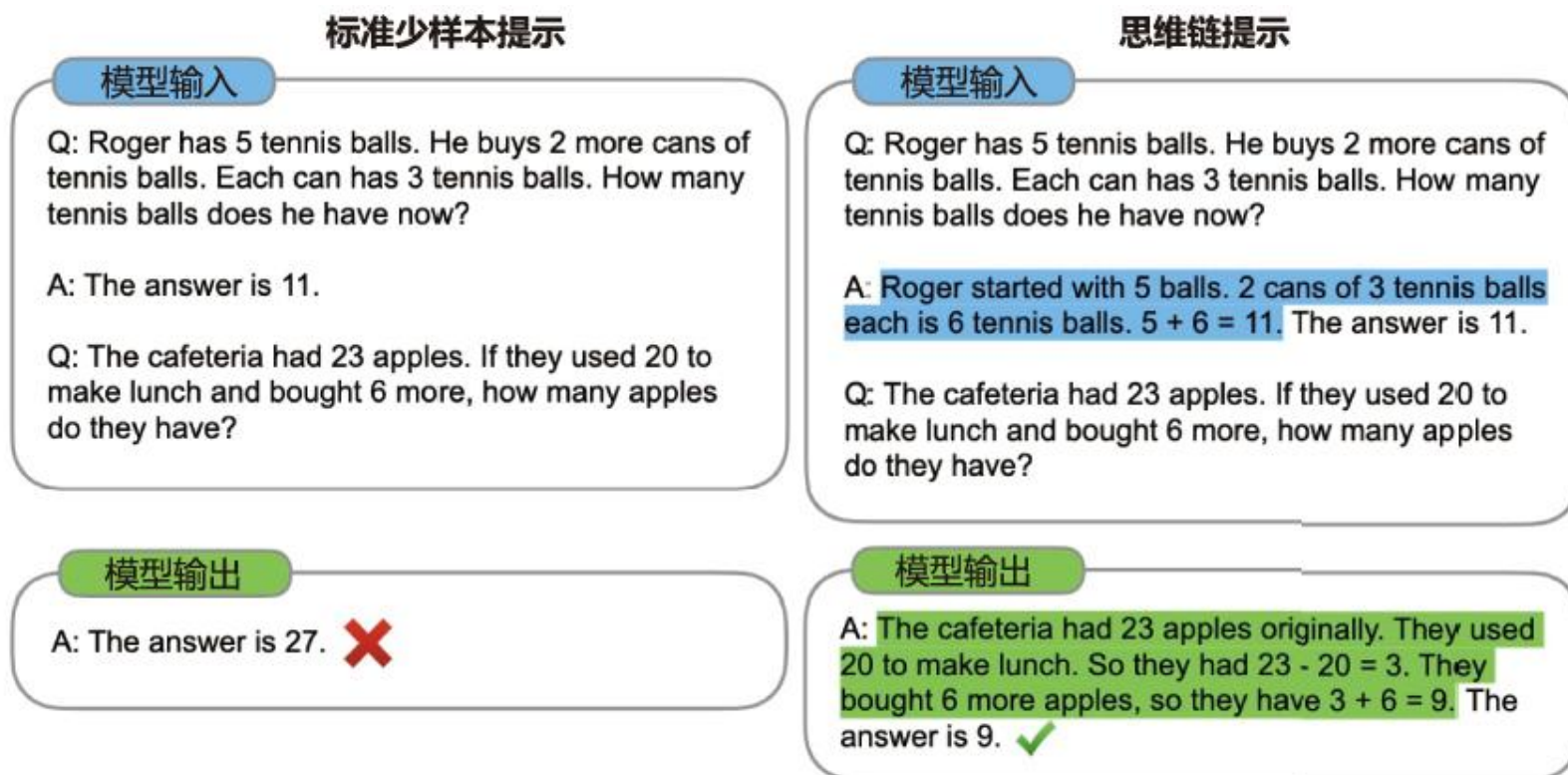


图 1.1: 思维链提示方式



当面对复杂任务或问题时，人类通常倾向于将其转化为多个更容易解决的子任务/子问题，并逐一解决它们，得到最终想要的答案或者结果。这种能力就是通常所说的**任务分解 (Task Decomposition)** 能力。基于这种问题解决思路，研究人員们提出了**由少至多提示 (Least-to-Most Prompting)** 方法。这种方法试图利用大语言模型的规划能力，将复杂问题分解为一系列的子问题并依次解决它们。

由少至多提示流程如图1.2 所示，主要包含两个阶段：**问题分解阶段**和**逐步解决子问题阶段**。在问题分解阶段中，模型的输入包括 $k \times [\text{原始问题}, \text{子问题列表}]$ 的组合，以及要测试的原始问题；在逐步解决子问题阶段中，模型的输入包括 $k \times [\text{原始问题}, m \times (\text{子问题}, \text{子答案})]$ 元组，以及要测试的原始问题和当前要解决的子问题。

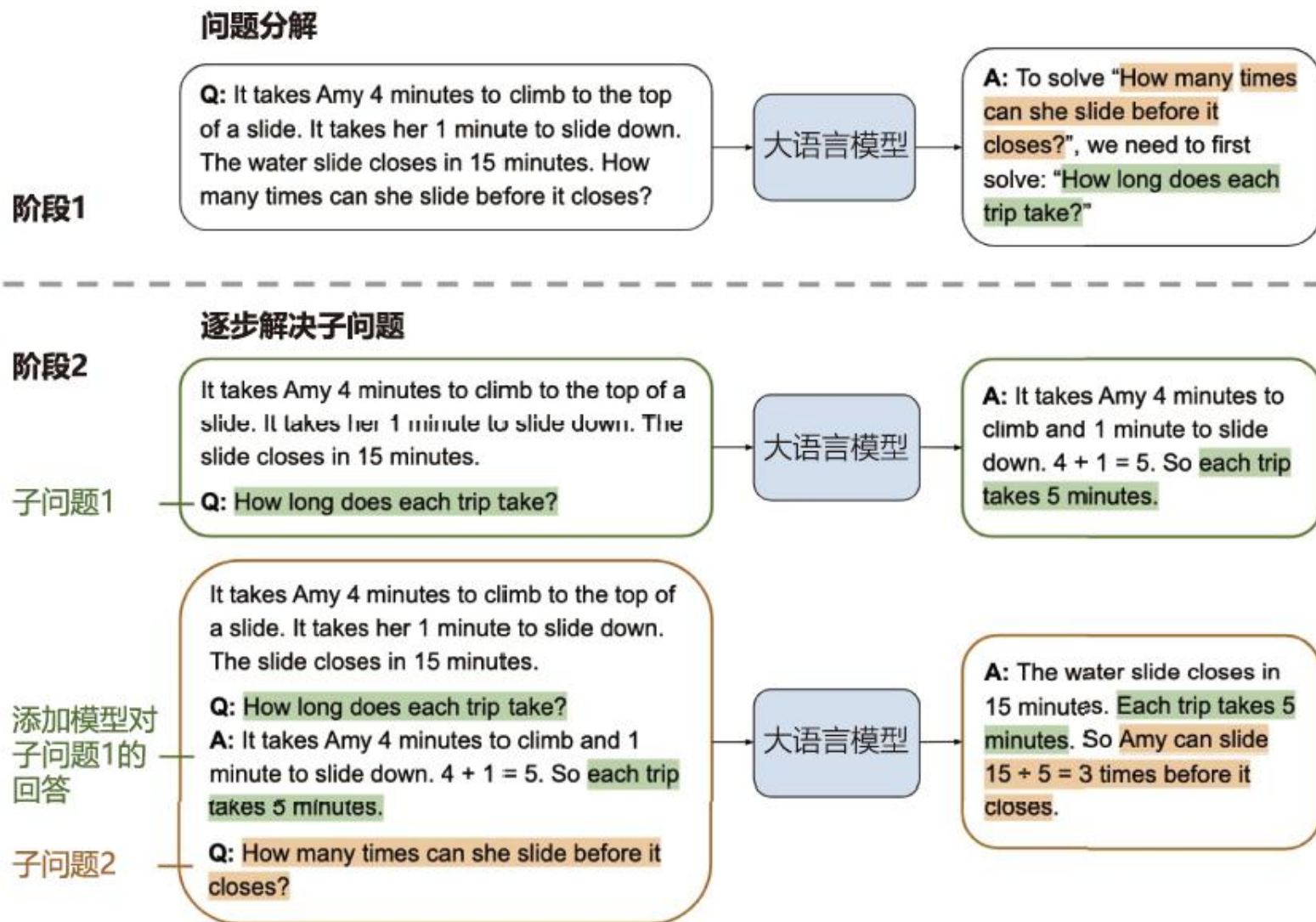


图 1.2: 由少到多提示流程

目录

Contents

1.1

推理规划

1.2

综合应用框架



ChatGPT 所取得的巨大成功，使得越来越多的开发者希望利用OpenAI 提供的API 或私有化模型开发基于大语言模型的应用程序。然而，即使大语言模型的调用相对简单，**仍需要完成大量的定制开发工作，包括API 集成、交互逻辑、数据存储等。**

为了解决这个问题，从2022年开始，多家机构和个人陆续推出了大量开源项目，帮助大家快速创建基于大语言模型的端到端应用程序或流程，其中较为著名的是LangChain 框架。

LangChain框架是一种利用大语言模型的能力开发各种下游应用的开源框架，旨在为各种大语言模型应用提供通用接口，简化大语言模型应用的开发难度。它可以实现数据感知和环境交互，即能够使语言模型与其他数据源连接起来，并允许语言模型与其环境进行交互。



使用LangChain 框架的核心目标是连接多种大语言模型（如ChatGPT、LLaMA 等）和外部资源（如Google、Wikipedia、Notion 及Wolfram 等），提供抽象组件和工具以在文本输入和输出之间进行接口处理。

大语言模型和组件通过“**链（Chain）**”连接，使得开发人员可以快速开发原型系统和应用程序。LangChain 的主要价值体现在以下几个方面。

(1) 组件化：LangChain 框架提供了用于处理大语言模型的抽象组件，以及每个抽象组件的一系列实现。这些组件具有模块化设计，易于使用，无论是否使用LangChain 框架的其他部分，都可以方便地使用这些组件。

(2) 现成的链式组装：LangChain 框架提供了一些现成的链式组装，用于完成特定的高级任务。这些现成的链式组装使得入门变得更加容易。对于更复杂的应用程序，LangChain 框架也支持自定义现有链式组装或构建新的链式组装。

(3) 简化开发难度：通过提供组件化和现成的链式组装，LangChain 框架可以大大简化大语言模型应用的开发难度。开发人员可以更专注于业务逻辑，而无须花费大量时间和精力处理底层技术细节。



LangChain 提供了以下6 种标准化、可扩展的接口，并且可以外部集成：

- 模型输入/输出 (Model I/O) ， 与大语言模型交互的接口；
- 检索 (Retrieval) ， 与特定应用程序的数据进行交互的接口；
- 链 (Chain) ， 用于复杂应用的调用序列；
- 记忆 (Memory) ， 用于在链的多次运行之间持久化应用程序状态；
- 智能体 (Agent) ， 语言模型作为推理器决定要执行的动作序列；
- 回调 (Callback) ， 用于记录和流式传输任何链式组装的中间步骤。



LangChain 中的模型**输入/输出 (Model I/O)** 模块是与各种大语言模型进行交互的基本组件，是大语言模型应用的核心元素。该模块的基本流程如图1.3 所示，主要包含以下部分：Prompts、Language Models 及Output Parsers。用户原始输入与模型和示例进行组合，然后输入大语言模型，再根据大语言模型的返回结果进行输出或者结构化处理。

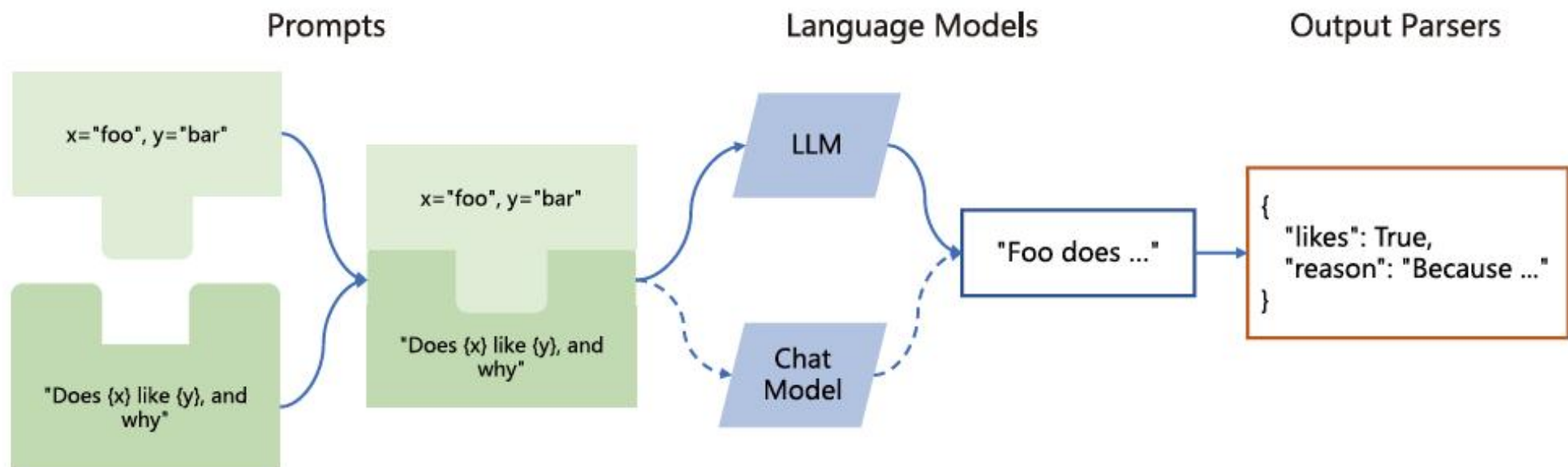


图 1.3: LangChain模型输入/输出基本流程

1.2.1 LangChain 框架核心模块--检索



许多大语言模型应用需要使用用户特定的数据，这些数据不是模型训练集的一部分。完成这一任务的主要方法是通过检索增强生成**Retrieval Augmented Generation (RAG)**，LangChain **数据连接 (Data connection)** 模块通过以下方式提供组件来加载、转换、存储和查询数据：Document loaders、Document transformers、Text embedding models、Vector stores 及 Retrievers。LangChain 数据连接模块的基本框架如图1.4 所示。

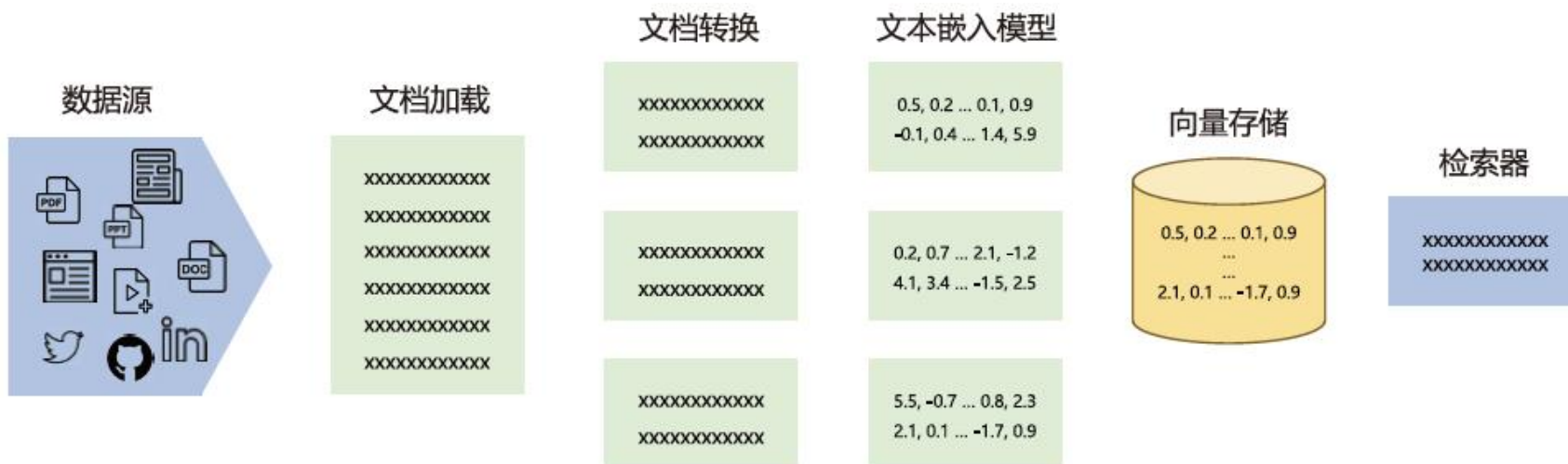


图 1.4: LangChain数据连接基本框架



虽然独立使用大语言模型能够应对一些简单任务，但对于更加复杂的需求，可能需要将多个大语言模型进行**链式（Chain）组合**。LangChain 为这种“链式”应用提供了Chain 接口，并将该接口定义得非常通用。**链允许将多个组件组合在一起，创建一个单一的、连贯的应用程序**。例如，可以创建一个链，接收用户输入，使用PromptTemplate 对其进行格式化，然后将格式化后的提示词传递给大语言模型。

```
from langchain.chat_models import ChatOpenAI
from langchain.prompts.chat import (
    ChatPromptTemplate,
    HumanMessagePromptTemplate,
)
human_message_prompt = HumanMessagePromptTemplate(
    prompt=PromptTemplate(
        template="What is a good name for a company that makes {product}?",
        input_variables=["product"],
    )
)
chat_prompt_template = ChatPromptTemplate.from_messages([human_message_prompt])
chat = ChatOpenAI(temperature=0.9)
chain = LLMChain(llm=chat, prompt=chat_prompt_template)
print(chain.run("colorful socks"))
```

1.2.1 LangChain 框架核心模块--记忆



大多数大语言模型应用都使用对话方式与用户交互。对话中的一个关键环节是能够引用和参考之前对话中的信息。对于对话系统来说，最基础的要求是能够直接访问一些过去的消息。在LangChain 中，这种能存储过去交互信息的能力被称为 **“记忆” (Memory)**。LangChain 记忆模块的基本框架如图1.5 所示。**记忆系统需要支持两个基本操作：读取和写入**。每个链都根据输入定义了核心执行逻辑，其中一些输入直接来自用户，但有些输入可以来源于记忆。

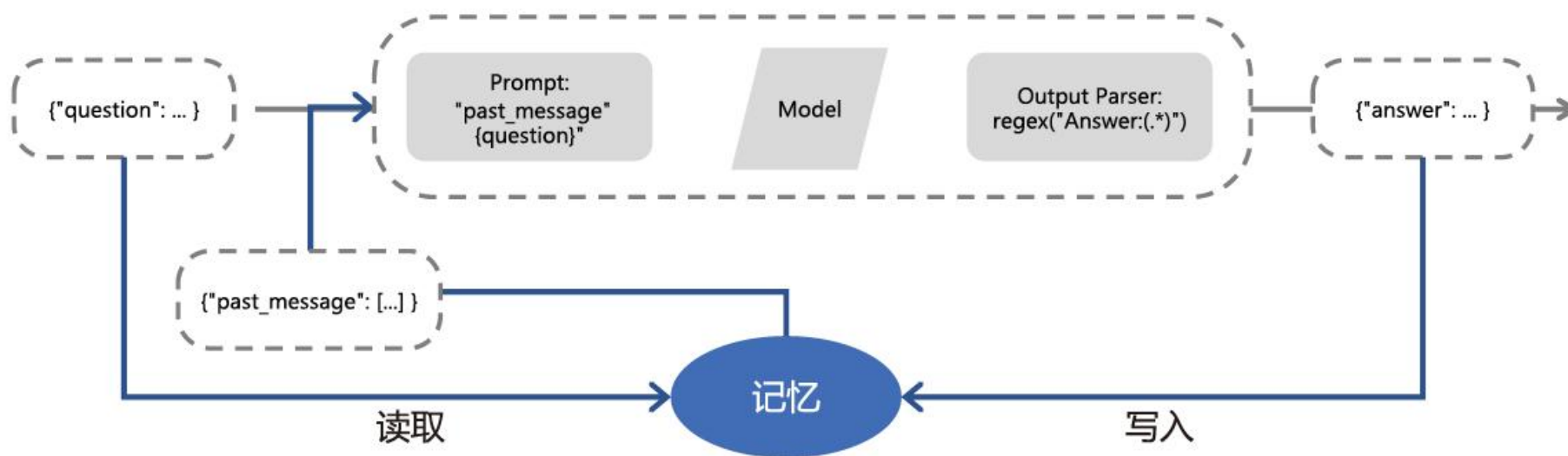


图 1.5: LangChain记忆模块基本框架



智能体 (Agent) 的核心思想是使用大语言模型来选择要执行的一系列动作。智能体通过将大语言模型与动作列表结合，选择最佳的动作序列，实现自动化决策和行动。

- **Agent:** 决定下一步该采取什么操作的类，由大语言模型和提示词驱动。提示词可以包括智能体的个性（有助于使其以某种方式做出回应）、智能体的背景上下文（有助于提供所要求完成的任务类型的更多上下文信息）、激发更好的推理的提示策略。
- **Tools:** 智能体调用的函数。这里有两个重要的考虑因素，一是为智能体提供正确的工具访问权限；二是用对智能体最有帮助的方式描述工具。
- **Toolkits:** 一组旨在一起使用以完成特定任务的工具集合，具有方便的加载方法。通常一个工具集合中有3 ~ 5 个工具。
- **AgentExecutor:** 智能体的运行空间，这是实际调用智能体并执行其所选操作的部分。除了 AgentExecutor 类，LangChain 还支持其他智能体运行空间，包括Plan-and-execute Agent、Baby AGI、Auto GPT 等。



例如，需要利用当前知识的用户输入，并给出如下调用方式：

```
agent_chain.run(input="whats the weather like in pomfret?")
```

给出如下回复：

```
> Entering new AgentExecutor chain...
{
  "action": "Current Search",
  "action_input": "weather in pomfret"
}
Observation: Cloudy with showers. Low around 55F. Winds S at 5 to 10 mph.
              Chance of rain 60%. Humidity76%.
Thought:{
  "action": "Final Answer",
  "action_input": "Cloudy with showers. Low around 55F. Winds S at 5 to 10 mph.
                  Chance of rain 60%. Humidity76%."
}

> Finished chain.

'Cloudy with showers. Low around 55F. Winds S at 5 to 10 mph. Chance of rain 60%. Humidity76%.'
```




LangChain 提供了**回调 (Callback)** 系统，允许连接到大语言模型应用程序的各个阶段。这对于日志记录、监控、流式处理和其他任务处理非常有用。可以通过使用 API 中提供的 `callbacks` 参数订阅这些事件。`CallbackHandlers` 是实现 `CallbackHandler` 接口的对象，每个事件都可以通过一个方法订阅。当事件被触发时，`CallbackManager` 会调用相应事件所对应的处理程序。

verbose 参数在 API 的大多数对象（`Chains`、`Models`、`Tools`、`Agents` 等）中都可用作构造函数参数，例如 `LLMChain(verbose=True)`，它等效于将 `ConsoleCallbackHandler` 传递给该对象及其所有子对象的 `callbacks` 参数。这对于调试非常有用，因为它会将所有事件记录到控制台。



大语言模型虽然可以很好地回答很多领域的各种问题，但是由于其知识是通过语言模型训练及指令微调等方式注入模型参数中的，**因此针对本地知识库中的内容，大语言模型很难通过此前的方式有效地进行学习**。通过LangChain 框架，则可以有效地融合本地知识库内容与大语言模型的知识问答能力。

基于LangChain 的知识库问答系统框架如图1.6 所示。知识库问答系统的工作流程主要包含以下几个步骤：

- (1) 收集领域知识数据构造知识库，这些数据应当能够尽可能地全面覆盖问答需求。
- (2) 对知识库中的非结构数据进行文本提取和文本分割，得到文本块。
- (3) 利用嵌入向量表示模型给出文本块的嵌入表示，并利用向量数据库进行保存。
- (4) 根据用户输入信息的嵌入表示，通过向量数据库检索得到最相关的文本片段，将提示词模板与用户提交问题及历史消息合并输入大语言模型。
- (5) 将大语言模型结果返回给用户。

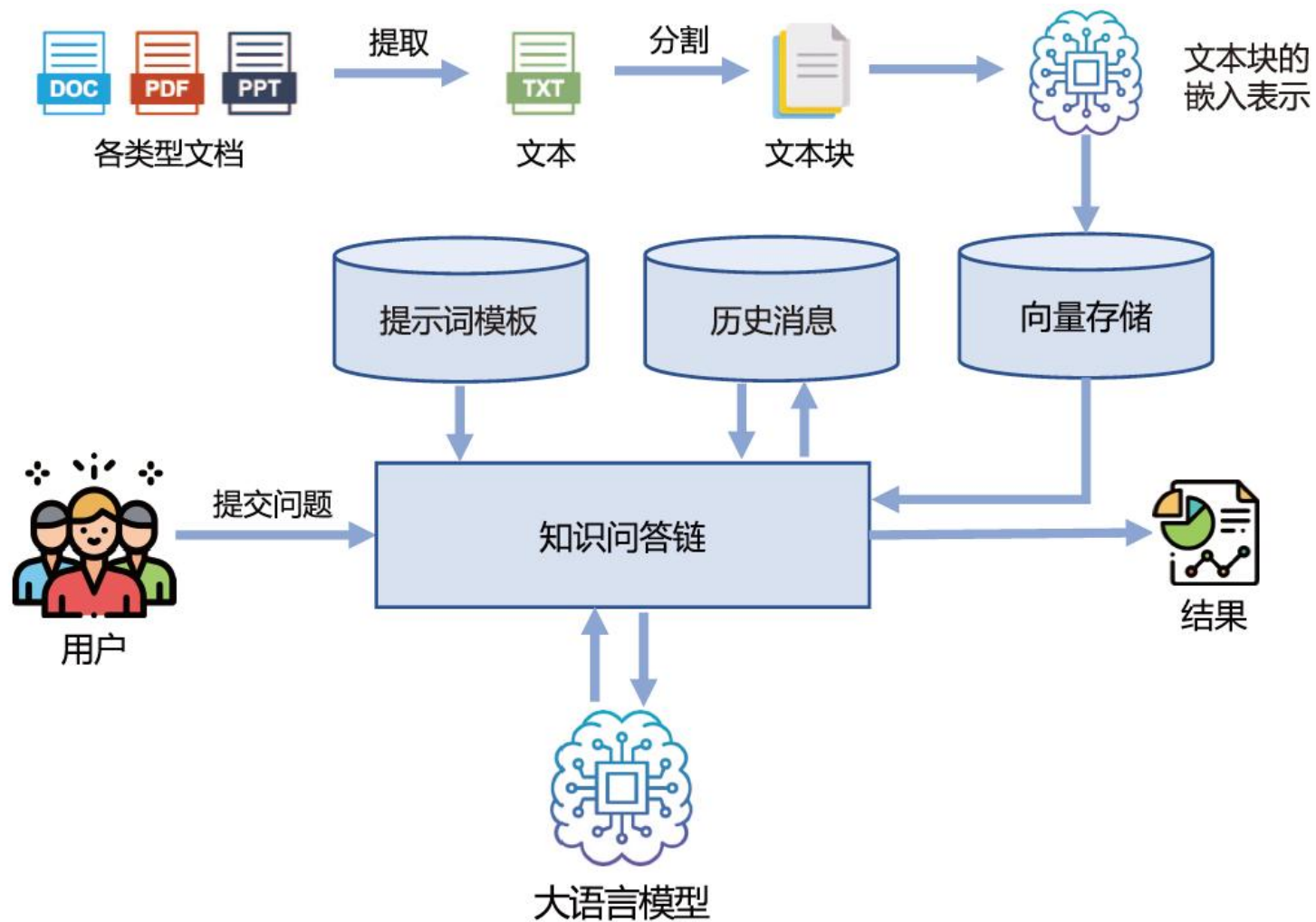


图 1.6: LangChain知识库问答框架



```
from langchain.document_loaders import DirectoryLoader
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import Chroma
from langchain.chains import ChatVectorDBChain, ConversationalRetrievalChain
from langchain.chat_models import ChatOpenAI
from langchain.chains import RetrievalQA

# 从本地读取相关数据
loader = DirectoryLoader(
    './Langchain/KnowledgeBase/', glob='**/*.pdf', show_progress=True
)
docs = loader.load()

# 将文本进行分割
text_splitter = CharacterTextSplitter(
```

```
    chunk_size=1000,
    chunk_overlap=0
)
docs_split = text_splitter.split_documents(docs)

# 初始化OpenAI Embeddings
embeddings = OpenAIEmbeddings()

# 将数据存入Chroma向量存储
vector_store = Chroma.from_documents(docs, embeddings)
# 初始化检索器，使用向量存储
retriever = vector_store.as_retriever()

system_template = """
Use the following pieces of context to answer the users question.
If you don't know the answer, just say that you don't know, don't try to make up an answer.
Answering these questions in Chinese.
-----
{question}
-----
{chat_history}
"""
```



```
# 构建初始消息列表
messages = [
    SystemMessagePromptTemplate.from_template(system_template),
    HumanMessagePromptTemplate.from_template('{question}')
]

# 初始化Prompt对象
prompt = ChatPromptTemplate.from_messages(messages)

# 初始化大语言模型，使用OpenAI API
llm=ChatOpenAI(temperature=0.1, max_tokens=2048)

# 初始化问答链
qa = ConversationalRetrievalChain.from_llm(llm,retriever,condense_question_prompt=prompt)

chat_history = []
while True:
    question = input(' 问题: ')
    # 开始发送问题chat_history为必须参数，用于存储历史消息
    result = qa({'question': question, 'chat_history': chat_history})
    chat_history.append((question, result['answer']))
    print(result['answer'])
```




- [1] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E.H., Xia, F., Le, Q., & Zhou, D. (2022). Chain of Thought Prompting Elicits Reasoning in Large Language Models. ArXiv, abs/2201.11903.
- [2] Zhou, D., Scharli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Bousquet, O., Le, Q., & Chi, E.H. (2022). Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. ArXiv, abs/2205.10625.