毕 业 设 计（论 文）

题 目 基于 LangChain 框架的个人智能助手

姓　　名 _____

学　　号 _____

所在学院 计算机学院

专业班级 _____

指导教师 _____

日　　期 2024 年 5 月 10 日

摘　要

自个人计算设备出现以来，个人智能助手（IPA）一直是研究人员和工程师关注的关键技术之一，它的出现旨在帮助用户高效获取信息和执行任务，为用户提供更智能、更方便、更丰富的交互体验。随着智能手机和物联网的发展，计算和传感设备变得无处不在，这极大地拓展了个人智能助手的功能边界。然而，由于缺乏用户意图理解、任务规划、工具使用和个人数据管理等能力，现有的个人智能助手的实用性和可扩展性仍然有限。

本文将设计并实现一种基于 LangChain 框架的个人智能助手，帮助用户减少日常生活中重复、乏味和低价值的劳动，让用户专注于更有趣、更有价值的事情，从而提高工作和生活的效率和质量。其使用多智能体技术将网络搜索、文本处理以及电子邮件管理等功能以无向图的形式结合在一起，使得用户可以通过聊天的形式更方便地处理上述事务。该个人智能助手还通过让多智能体扮演不同职能的模式组建一个研究团队，从而赋予这款个人智能助手具备产品调研的能力。经过实验与测试，本文实现的个人智能助手能够切实处理连续任务，代替用户处理繁琐事务。

关键词：LangChain；大语言模型；提示工程；智能体；个人智能助手

Abstract

Since the emergence of personal computing devices, personal intelligent assistants (IPAs) have been one of the key technologies that researchers and engineers have been focusing on, emerging to help users efficiently access information and perform tasks, and to provide a sma rter, more convenient, and richer interaction experience. With the development of smartphone s and the Internet of Things (IoT), computing and sensing devices have become ubiquitous, w hich greatly expands the functional boundaries of personal intelligent assistants. However, the utility and scalability of existing personal intelligent assistants are still limited due to the lack of capabilities such as user intent understanding, task planning, tool usage, and personal data management.

In this paper, we will design and implement a personal intelligent assistant based on the LangChain framework to help users reduce repetitive, tedious and low-value labor in their daily lives and allow them to focus on more interesting and valuable things, thus improving the efficiency and quality of their work and life. It uses multi-intelligence technology to combine functions such as web searching, text processing, and email management in the form of a directionless graph, making it easier for users to deal with the aforementioned matters through chat. The personal intelligent assistant also empowers this personal intelligent assistant with the ability to conduct product research by forming a research team through the model of having multiple intelligences play different functions. After experimentation and testing, the personal intelligent assistant realized in this paper is able to practically handle continuous tasks and deal with tedious matters instead of the user.

Keywords: LangChain; Large Language Model; prompt engineering; agents; personal intelligent assistant

目 录

第 1 章 绪论

1.1 研究背景与意义

随着智能手机、智能家居设备、电动汽车等个人设备的广泛普及和机器学习技术的进步，很多移动设备都内嵌了个人助手软件，比如 Siri[1]、Google Assistant[2]、Alexa[3] 等。这些个人助手与用户密切相关，可以读取用户数据和传感器数据、控制各种个人设备、使用与个人账户关联的个性化服务。但是，这些个人助手的灵活性和可扩展性都还有限。它们的智能水平还远远不够，在理解用户意图、推理和任务执行等方面尤其明显。现如今大多数个人助手都只能执行受限范围内的任务（比如内置应用的简单功能）。一旦用户的任务请求超出了这些范围，它们就无法准确理解和执行这些动作。

近年来，以大语言模型（LLM）为代表的预训练大型基础模型得到了快速发展，为个人助手领域带来了新的机遇。受大语言模型能力的启发，研究人员试图让大语言模型自主使用工具来完成复杂的任务。例如，例如控制用于信息检索和摘要的浏览器，调用用于机器人行为控制的机器人编程接口，以及调用用于复杂数据处理的代码解释器等。将这些功能集成到个人助理中是一个自然的想法，从而实现更智能的方式来操作个人数据、个人设备和个性化服务。

因此，个人智能助手定义为一种特殊类型的基于大语言模型的个人助手，通过大语言模型的能力赋予个人助手更加“智能”的行为，它与个人数据、个人设备和个人服务深度集成。其主要目的是帮助用户，帮助减少重复和繁琐的工作，并更多地关注有趣和重要的事务。个人智能助手的出现为人们在处理繁琐、重复的工作时提供了新思路，使得他们不再需要投入过多的时间和精力，而是可以更加专注于那些更加重要的任务，从而可以使得工作效率得到极大的提升。

1.2 国内外研究现状

个人助手有着悠久的发展历史，发展过程可分为四个阶段，如下图 1-1 所示。下面将具体介绍这四个阶段。

第一阶段从 20 世纪 50 年代到 80 年代末，主要是语音识别技术的发展。语音识别的早期阶段是从基本的数字和单词开始的。贝尔实验室开发了“Audrey”，它可以识别数字

0-9，准确率约为 90%。1962 年，IBM 高级系统开发部实验室推出了“shoebox”[4]系统，该系统能够识别多达 16 个单词。从 1971 年到 1976 年，由美国国防部资助的语音理解研究项目显著提高了语音识别技术。Harpy 系统[5]尤其具有代表性，因为它可以理解由 1011 个单词组成的句子，相当于一个三岁孩子的熟练程度。1986 年，IBM 开发了 Tangora 语音识别打字系统，能够识别 20000 个单词，并提供预测和纠错功能。Tangora 系统利用了隐马尔可夫模型[6]，需要对单个说话者进行语音识别训练，每个单词之间都有停顿。

第二阶段涵盖了从 20 世纪 90 年代到 21 世纪末的这段时间，因为语音识别开始集成到用于某些高级功能的软件中。1990 年，“Dragon Dictate”软件[7]发布，这是第一款面向消费者的语音识别产品。它最初设计用于 Microsoft Windows，支持离散语音识别。“Speakable items”[8]由苹果公司于 1993 年推出，使用户能够用自然的语言控制电脑。1996 年，IBM 为放射科医生推出了“MedSpeak”[9]，这也是第一个支持连续语音识别的商业产品。微软于 2002 年将语音识别集成到 Office 应用程序中，谷歌于 2008 年在 iPhone 上的谷歌移动应用程序中添加了语音搜索。

第三阶段从 2010 年代初开始。在这一时期，一直在线的虚拟助理服务开始出现在智能手机和个人电脑等移动设备上。Siri 被广泛认为是第一款安装在现代智能手机上的智能个人助理，2011 年被集成到苹果的 iPhone 4S 中。自推出以来，Siri 一直是苹果设备的关键内置软件，包括 iPhone、iPad、Apple Watch、HomePod 和 Mac，并不断进行更新和迭代，融入新功能。与 Siri 类似，许多其他虚拟智能助理也在这一时期开始出现。2014 年，微软发布了 Cortana[10]，并逐步将其集成到台式机等平台中。亚马逊在同年发布了 Alexa，它可以完成语音交互、音乐播放、设置警报等任务。除了语音搜索，谷歌助手于 2016 年推出，支持用户通过语音和键盘输入进行交互。

第四阶段最近开始，大语言模型开始引起世界各地的关注。基于大语言模型，出现了许多智能聊天机器人（例如，ChatGPT[11]），以及一些安装在个人设备上大语言模型赋能的个人助手软件（例如，Copilot[12]）。

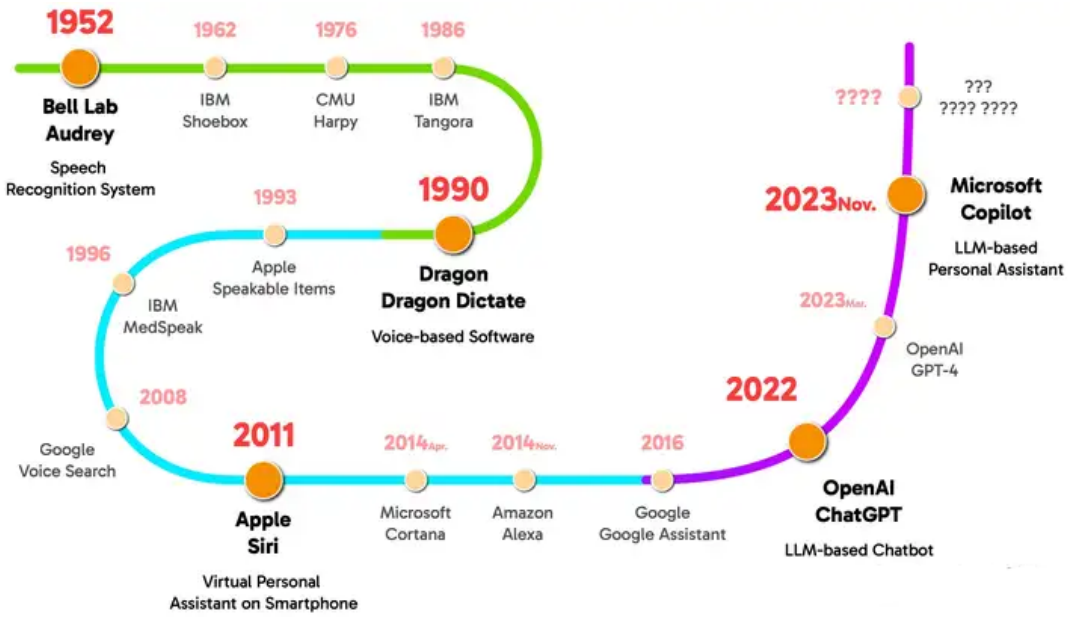

图 1-1 个人助手发展阶段[3]

在大语言模型出现之前的很长一段时间内，个人助手的灵活性和可扩展性都还有限。它们的智能水平还远远不够，在理解用户意图、推理和任务执行等方面尤其明显。如今大多数个人助手同样也只能执行受限范围内的任务（比如内置应用的简单功能）。一旦用户的任务请求超出了这些范围，智能体就无法准确理解和执行这些动作。

要改变这种情况，就必须显著提升个人助手的能力，使其支持范围更广、更灵活的任务。但是，当前的个人助手产品很难支持大范围的任务，大多数都需要遵循特定的预定义规则，比如开发者定义的规则或用户演示的步骤。除了定义任务执行的触发器和步骤之外，开发者或用户还必须明确指定他们希望支持哪些功能。本质上讲，这种方法会限制这些应用被用于更广泛的任务，因为支持更多任务需要大量时间和劳动力成本。有些方法在尝试通过监督学习或强化学习实现自动化学习上下工夫，希望支持更多任务。但是，这些方法也需要大量人工演示和或定义奖励函数的工作。

大语言模型的出现为个人助手领域带来了新的可能性，语言模型的规模化定律揭示了增加模型参数对提高模型性能的重要性。大语言模型通常以无监督的方式使用大规模开放域文本数据进行训练，然后进行指令微调[13]和人类反馈强化学习（RLHF）[14]，提高性能和一致性。OpenAI 于 2022 年底推出的 ChatGPT 是大语言模型的一个里程碑，展示了惊人的问答能力。通过将简单的任务描述作为输入提示输入到大语言模型中，大语言模型的任务和响应可以很容易地进行自定义。此外，这些模型还展示了在各种语言理解和推理任务中强大的泛化能力。ChatGPT 本身可以被视为一种智能个人助理，在文本

响应中返回信息来帮助用户。

个人智能助手将成为大语言模型时代个人设备的主要软件范例。然而，目前这一研究方向目前仍处于早期探索阶段，个人智能助手的软件堆栈和生态系统仍处于非常早期的阶段，许多与系统设计和实施相关的重要问题尚不清楚。尽管个人智能助手显示出巨大的潜力，但距离真正理解并使用个人智能助手帮助用户的最终目标还有相当大的距离。许多与效率、安全和隐私相关的问题还没有得到充分解决。

在国外，已经有一些商业产品尝试将大语言模型与个人助手集成。例如，微软的 Copilot 系统集成了 GPT-4[15]的功能，帮助 Windows 用户自动起草文档、创建演示文稿、总结电子邮件，从而提高了用户的工作效率。新必应[16]还改善了上网体验，提供了一个强大高效的搜索引擎，可以更好地了解用户的需求。同样，谷歌将大语言模型（Bard[17]，Gemini[18]）集成到搜索引擎中，以实现更方便的网络搜索体验。

在国内，华为、小米、Oppo、Vivo 在内的智能手机公司也将大语言模型（PanGu[19]、MiLM[20]等）集成到其设备产品中。其中的一些采用了基于本地部署的轻量级大语言模型解决方案。到目前为止，这些商业产品大多只是将大语言模型的聊天界面简单地集成到个人助理中。

1.3 主要研究内容

目前，国内外许多公司都在积极探索和研发适合自身需求的个人智能助手。尽管这些助手在特定领域内表现出色，但在综合办公环境下，尚未有一款产品能够真正实现对繁杂事务的简化与整合。针对这一问题，本文提出了基于 LangChain 框架[21]的个人智能助手设计方案。该方案旨在通过整合网络搜索、文本处理及邮件管理等功能，为用户提供一站式的办公服务。帮助用户在工作过程中简化各类繁琐操作。希望通过将这些看似独立的任务进行整合，从而使广大用户得以从这些繁杂的工作中解脱出来，将更多的精力投注到更为关键的工作环节中去。

本文将通过对现有个人智能助手的发展现状进行深入剖析，将结合实际案例，验证基于 LangChain 框架的个人智能助手设计方案的可行性和实用性。本文将着重关注以下几个典型应用场景，通过对这些场景中所涉及的具体任务进行分析与解决，以充分展示本文所构建的个人智能助理所具有的强大功能。

例如，当企业管理者计划在周末组织团队建设活动时，他需要先通过网络搜索了解

次日的天气状况，随后撰写相应的电子邮件分发给员工。本文实现的个人智能助手将简化该流程，将这些步骤环节通过一句简单的对话或指令即可完成。

再比如，员工在工作过程中经常会遇到大量的邮件，需要对这些邮件进行整理，提取电子邮件中的重要信息，并将之整理。这时，本文介绍的个人智能助手将帮助完成读取文件的任务，并将其写入到文本文件中，让用户可以更加轻松的查看这些邮件。

又或者，当员工需要针对上级领导提出的某项问题展开深入研究时。本个人助手将对该话题进行研究，通过网络搜索收集资料，最终生成一份研究报告提供给用户查看。

1.4 论文结构安排

第一章作为本文绪论部分，主要从课题的角度出发，深入探讨了由大语言模型赋能的个人智能助手的理论基础、发展背景及其在现代社会中的实际价值与意义，同时也参照国际视野，对这一领域的研究进展状况作了详尽综述。在此之后，我们简要概述了论文中所涉及到的主要科研方向及研究内容，及文章的整体结构布局。

第二章以丰富的实例为依托，对个人智能助手开发过程中所涉及的关键性知识进行了详细阐述，包括大语言模型的结构、大语言模型训练流程、模型微调方法、提示工程和智能体等内容。

第三章则是对 LangChain 框架进行全方位的剖析，详细解读了该框架中各个重要组成部分的功能特性，并对其在整个 LangChain 生态系统中的作用进行了简要介绍。

第四章将介绍如何利用 LangChain 框架来构建个性化的个人智能助手，它的整体结构设计，以及对其表现进行展示评估。

第五章为总结与展望，在该章节将对全篇论文进行简要的总结，指出本文工作的不足之处，并对未来可改进或研究的地方进行展望。

第 2 章 相关技术

基于 LangChain 框架的个人智能助手依托智能体，智能体技术由大语言模型作为基础，通过提示工程中的 ReAct[22]框架，使得模型可以使用各种工具。提示工程是一种微调模型的技术，在不改变模型的基础之上，增强模型的回复能力。本章将对相关技术进行阐述。

2.1 大语言模型

大语言模型是指那些拥有大规模参数及复杂计算结构的机器学习模型。这些模型常由深度神经网络所构建，包含了数十亿乃至数千亿个参数。大语言模型的设计初衷在于提升模型的表达能力与预测性能，使其能够应对更为复杂的任务和数据。这类模型通常是具备大规模参数和计算能力的自然语言处理模型，如 OpenAI 的 GPT-3 模型。这些模型可通过大量的数据和参数进行训练，从而生成与人相似的文本或解答自然语言的问题。大型语言模型在自然语言处理、文本生成以及智能对话等领域有着广泛的应用。

ChatGPT 对于大语言模型的解释更为深入浅出，同时也更能体现出类人的归纳和思考能力：大语言模型本质上是一个利用海量数据进行训练的深度神经网络模型，其庞大的数据和参数规模，实现了智能的涌现，展现出类人的智能。

2.1.1 大语言模型结构简述

2018 年，自然语言处理（NLP）领域也步入了大语言模型时代，谷歌出品的 Bert[23]模型横空出世，碾压了以往的所有模型，直接在各种自然语言处理的建模任务中取得了最佳的成绩。与此同时，OpenAI 早于 Bert 出品了一个初代 GPT 模型。它们大致思想是一样的。都基于 Transformer 这种编码器，获取了文本内部的相互联系。两者最主要的区别在于，Bert 仅仅使用了编码器部分进行模型训练，GPT 仅仅使用了 decoder 部分。

GPT 的核心是 Transformer[33]架构，它是一种深度学习模型，主要用于处理自然语言处理任务，如语言翻译、文本生成等。Transformer 模型由多个自注意力层（Self-Attention Layer）和前馈神经网络层（Feed-Forward Neural Network Layer）组成，其结构如图 2-1 所示，各个层的信息如下：

(1) 自注意力层。该层是 Transformer 模型的核心部分，它通过计算输入序列中每个

位置上的词向量之间的相似度来生成每个位置的表示向量。具体来说，自注意力层会对输入序列中的每个词向量进行加权平均，生成一个全局表示向量，这个向量可以捕捉到输入序列中的语义信息。

(2) 前馈神经网络层。该层是一种全连接神经网络，它将自注意力层的输出向量进行进一步的处理，生成每个位置的输出向量。在前馈神经网络层中，每个位置的输出向量是由前一层中的隐藏状态和当前位置的输入向量共同决定的。

(3) Transformer 编码器。该编码器是由多个自注意力层和前馈神经网络层交替堆叠而成的。它将输入序列分成若干个长度相等的块，并对每个块进行独立的编码。编码器的输出向量是所有块中对应位置的向量拼接而成。

(4) Transformer 解码器。该解码器也是由多个自注意力层和前馈神经网络层交替堆叠而成的。它接收编码器的输出向量作为输入，并生成输出序列。在解码过程中，解码器会使用编码器输出的上下文向量（Context Vector）来指导输出序列的生成。

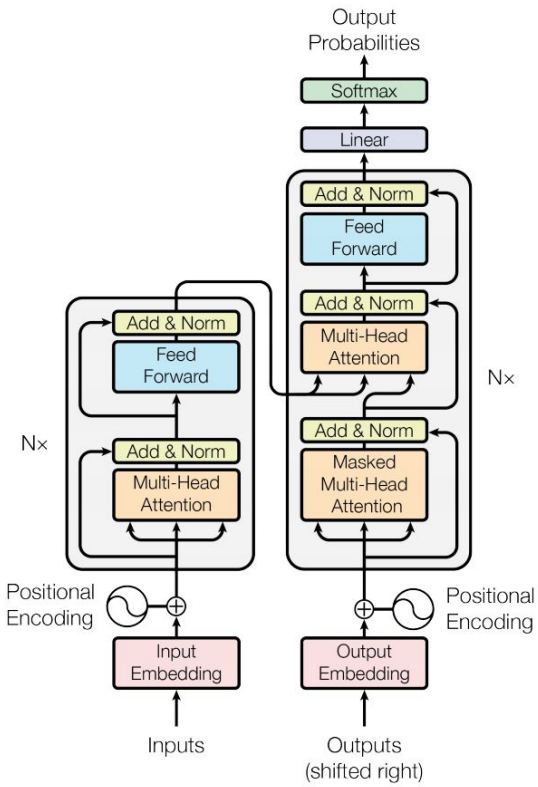

图 2-1 Transformer 结构[33]

2.1.2 训练流程

(1) 预训练阶段

大语言模型在大规模语料库上进行预训练，大语言模型可以获得基本的语言理解和生成能力。预数据用有通用文本和专用文本，预训练为大语言模型的能力奠定了基础。

通用文本包含：

1) 网页，如 CommonCrawl，数据量大，使得大语言模型能够获得多样化的语言知识并增强大语言模型的泛化能力。
2) 对话文本，如公共对话语料，对话数据可以增强大语言模型的对话能力 ，能有效改善大语言模型在问答任务上的表现。
3) 书集，如 Books3，提供更加正式的文本，语言知识建模长期依赖关系以及连贯文本能力

专用文本包括：

1) 多语言文本：整合多种语言的语料库，能够增强多语言的生成和理解能力。
2) 科学文本：如论文、教材，使能大语言模型在科学和推理任务中取得良好性能。
3) 代码：显著提升代码编写能力，且可能是复杂推理能力链式思考（CoT）的来源。

(2) 微调阶段

数据用于构建特定的微调数据集，在预训练阶段之后，大语言模型可以获得解决各种任务的通用能力。但是为了使大语言模型拥有解决特定问题的能力，这时就需要对大语言模型进行微调，进一步适配到特定的任务。大语言模型微调（Fine-tuning）是指在已经预训练好的大型语言模型基础上，使用特定的数据集进行进一步的训练，以使模型适应特定任务或领域。其根本原理在于，机器学习模型只能够代表它所接收到的数据集的逻辑和理解，而对于其没有获得的数据样本，其并不能很好地识别或理解，对于大语言模型而言，其同样也无法很好地回答特定场景下的问题。

预训练阶段完成后得到的预训练模型（Pre-trained Model），已经可以完成很多任务，比如回答问题、总结数据、编写代码等。但是，并没有一个模型可以解决所有的问题，尤其是行业内的专业问答、关于某个组织自身的信息等，是通用大语言模型所无法触及的。在这种情况下，就需要使用特定的数据集，对合适的基础模型进行微调，以完成特定的任务、回答特定的问题等。在这种情况下，微调就成了重要的手段。例如，布洛芬到底能否和感冒药同时吃？为了确定模型可以回答正确，我们就需要对基础模型进行微调。

根据微调对整个预训练模型的调整程度，微调可以分为全微调和重用两个方法，根据微调使用的数据集的类型，大语言模型微调还可以分为监督微调和无监督微调两种，具体大语言模型微调方法将在下一小节中详细说明。

(3) 推理阶段

在预训练或适应性调整之后，使用大语言模型的主要方法是为解决各种任务设计适合的提示策略。设计良好的提示对于揭示大语言模型完成特定任务的能力非常有帮助。提示词设计有以下几个原则：清晰明确地表达任务目标、将任务分解为易懂的子任务、利用模型友好的格式。提示工程需要考虑关键要素 包括任务描述、输入数据、上下文信息、提示风格以及示例，以下是这些关键要素：

1) 任务描述。应该用自然语言清楚地描述任务目标。对于具有特殊输入或输出格式的任务，通常需要详细澄清，并可以利用关键字来突出显示特殊设置，以更好地指导大语言模型完成任务。
2) 输入数据。在大多数情况下 ，可以用自然语言描述输入数据 ，例如，需要大语言模型回答的实例)。对于特殊输入数据，如知识图谱和表格、数据库等结构化数据，可以采用适当且方便的方法 API 使它们对大语言模型可读。
3) 上下文信息。除了任务描述和输入数据，上下文或背景信息对于特定任务也是必不可少的。例如，文档检索（retrieved documents）对于开放领域问题回答作为支持证据非常有用。上下文任务描述对于引导大语言模型完成复杂任务也很有帮助，可以更好地描绘任务目标，特殊输出格式以及输入和输出之间的映射关系。
4) 提示风格。对于不同的大语言模型，设计适合的提示风格以揭示它们解决特定任务的能力是很重要的。应该将提示表达为一个清晰的问题或详细的指令 ，可以被很好地理解和回答。在某些情况下，添加前缀或后缀可以更好地指大语言模型。例如，使用前缀“让我们逐步思考”可以帮助大语言模型进行逐步推理，使用前缀“你是这个任务（或这个领域）的专家”可以提高大语言模型在某些特定任务中的性能。
5) 示例。大语言模型可以从上下文学习中受益，以解决复杂任务，其中提示包含一小部分所需输入-输出对的任务示例，即少而精的演示。在实践中，建议为目标任务生成几个高质量的演示，这将极大地提升任务性能。

2.1.3 大语言模型微调

大语言模型的微调技术可分为如下两种全面微调和参数高效微调。

全面微调（Fine-tuning）。涉及调整所有层和参数，以适配特定任务。此过程通常采用较小的学习率和特定任务的数据，可以充分利用预训练模型的通用特征，但可能需要更多计算资源。

参数高效微调（Parameter-Efficient Fine-Tuning，PEFT）[34]。旨在通过最小化微调参数数量和计算复杂度，提升预训练模型在新任务上的表现，从而减轻大型预训练模型的训练负担。即使在计算资源受限的情况下，参数高效微调技术也能够利用预训练模型的知识快速适应新任务，实现有效的迁移学习。因此，参数高效微调不仅能提升模型效果，还能显著缩短训练时间和计算成本，使更多研究者能够参与到深度学习的研究中。参数高效微调包括 LoRA、QLoRA、适配器调整(Adapter Tuning)、前缀调整(Prefix Tuning)、提示调整(Prompt Tuning)、P-Tuning 及 P-Tuning v2 等多种方法，这些方法如图 2-2 所示。

本文实现的智能助手使用提示工程作为底层原理，而提示工程本质是提示调整，所以在下文仅对提示调整进行概述。

提示调整是一种在预训练语言模型输入中引入可学习嵌入向量作为提示的微调方法。这些可训练的提示向量在训练过程中更新，以指导模型输出更适合特定任务的响应。提示调整与前缀调整都涉及在输入数据中添加可学习的向量，这些向量是在输入层添加的，但两者的策略和目的不同：

(1) 提示调整。旨在模仿自然语言中的提示形式，将可学习向量（通常称为提示标记）设计为模型针对特定任务生成特定类型输出的引导。这些向量通常被视为任务指导信息的一部分，倾向于使用较少的向量来模仿传统的自然语言提示。

(2) 前缀调整。可学习前缀更多地用于提供输入数据的直接上下文信息，作为模型内部表示的一部分，可以影响整个模型的行为。

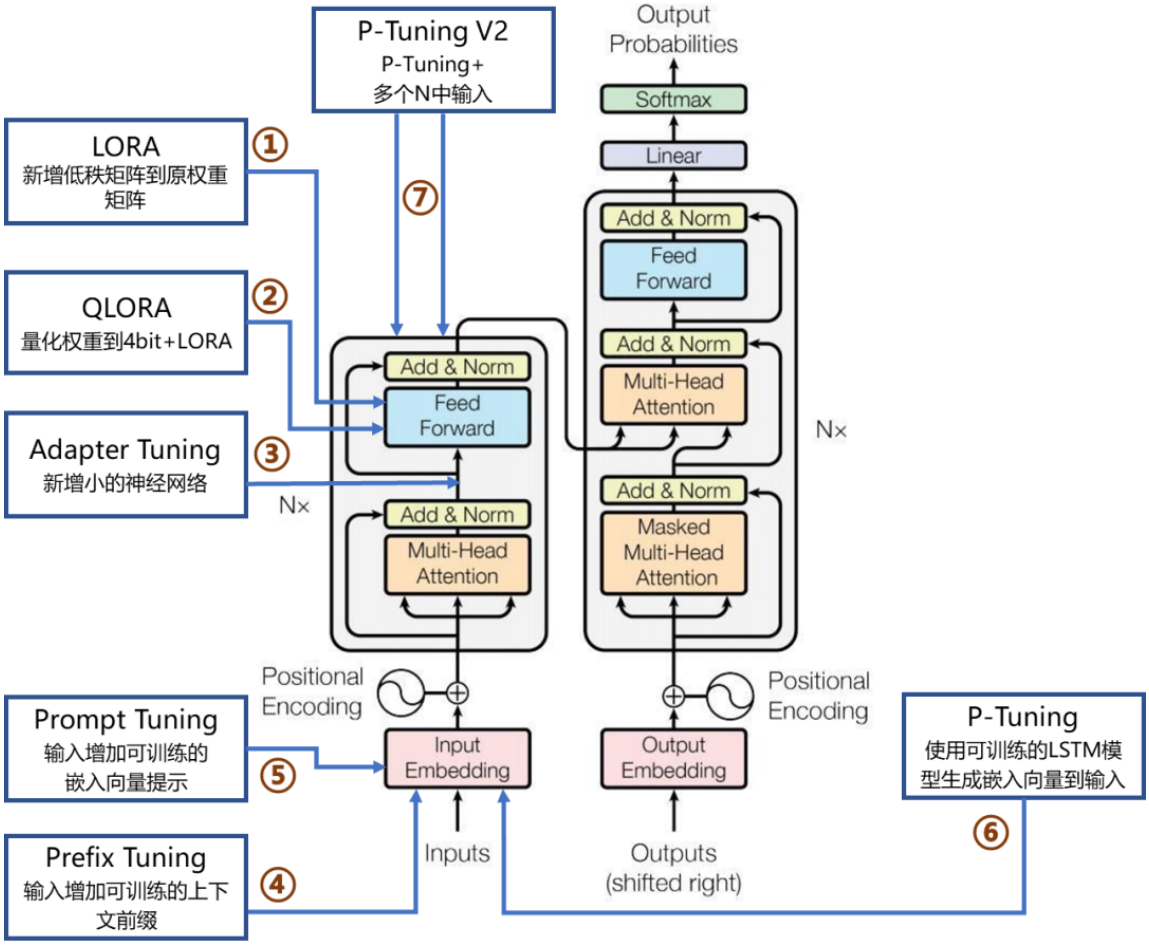

图 2-2 大语言模型微调方法[34]

2.1.4 大语言模型参数设置

针对普通用户，上一小节提到的在微调阶段对预训练模型进使用特殊数据集进行训练开销花费较大，此种方法对于普通用户来说还是尤为困难，本节将介绍在模型微调阶段所进行的更为简单便利的微调方式，即通过大语言模型启动参数进行小幅度的调整，但该方法对于大语言模型效果提升较为有限。

不同于传统深度学习网络，大语言模型很难通过更改网络结构进行修改，也很难重新进行训练。在大语言模型应用中用户一般会通过应用程序结构（API）或直接与大语言模型进行交互。若用户需要对大语言模型进行一定程度的自定义，可以通过配置一些参数以获得不同的提示结果。调整这些设置对于提高响应的可靠性非常重要，用户可能需要进行一些实验才能找出适合您的用例的正确设置。以下是使用不同大语言模型提供程序时会遇到的常见设置：

Temperature：简单来说，temperature 的参数值越小，模型就会返回越确定的一个结果。如果调高该参数值，大语言模型可能会返回更随机的结果，也就是说这可能会带来更多样化或更具创造性的产出。在实际应用方面，对于需要质量保障的任务，例如知识库问答等，用户可以设置更低的 temperature 值，以促使模型基于事实返回更真实和简洁的结果。对于诗歌生成或其他创造性任务，用户可以适当调高 temperature 参数值。

(1) Top_p。使用 top_p（与 temperature 一起称为核采样的技术），可以用来控制模型返回结果的真实性。如果用户需要准确和事实的答案，就把参数值调低，想要更多样化的答案，就把参数值调高一些。一般建议是改变 Temperature 和 Top P 其中一个参数就行，不用两个都进行调整。

(2) Max Length。用户可以通过调整 max length 来控制大语言模型生成的 token 数。指定 Max Length 有助于防止大语言模型生成冗长或不相关的响应并控制成本。

(3) Stop Sequences。stop sequence 是一个字符串，可以阻止模型生成 token，指定 stop sequences 是控制大语言模型响应长度和结构的另一种方法。例如，用户可以通过添加“11”作为 stop sequence 来告诉模型生成不超过 10 个项的列表。

(4) Frequency Penalty。frequency penalty 是对下一个生成的 token 进行惩罚，这个惩罚和 token 在响应和提示中出现的次数成比例，frequency penalty 越高，某个词再次出现的可能性就越小，这个设置通过给 重复数量多的 token 设置更高的惩罚来减少响应中单词的重复。

(5) Presence Penalty。presence penalty 也是对重复的 token 施加惩罚，但与 frequency penalty 不同的是，惩罚对于所有重复 token 都是相同的。出现两次的 token 和出现 10 次的 token 会受到相同的惩罚。此设置可防止模型在响应中过于频繁地生成重复的词。如果希望模型生成多样化或创造性的文本，可以设置更高的 presence penalty。如果希望模型生成更专注的内容，可以设置更低的 presence penalty。

与 temperature 和 top_p 一样，一般建议是改变 frequency penalty 和 presence penalty 其中一个参数就行，不要同时调整两个。最终生成的结果可能会和使用的大语言模型的版本而异。

2.2 提示工程

提示工程（Prompt engineering）是人工智能中的一个概念，特别是自然语言处理

（NLP）。在提示工程中，任务的描述会被嵌入到输入中。例如，不是隐含地给予模型一定的参数，而是以问题的形式与用户的提问同时输入。其典型工作方式是将一个或多个任务转换为基于提示的数据集，并通过所谓的“基于提示的学习（prompt-based learning）”来训练语言模型。它也可以从一个大型的“冻结”预训练语言模型开始工作，其中只学习了提示的表示方法，即所谓的“前缀调整（prefix-tuning）”或“提示调整（prompt tuning）”。语言模型 GPT-2 和 GPT-3 是提示工程的重要步骤。

2.2.1 提示词使用方法

用户可以通过简单的提示词获得大量结果，但结果的质量与用户提供的信息数量和完善度有关。一个提示词可以包含用户传递到模型的“指令”或“问题”等信息，也可以包含其他详细信息，如“上下文”、“输入”或“示例”等。用户可以通过这些元素来更好地指导模型，并因此获得更好的结果。

如果使用的是 OpenAI 操作面板或者其他具有该功能的任何大语言模型的操作面板，则可以以可视化的形式使用提示词以提示模型，如图 2-3 所示，用户可以在操控面板的正中心内编写需要的提示词，而右边的模型调整栏正是上文中提到的模型参数设置。

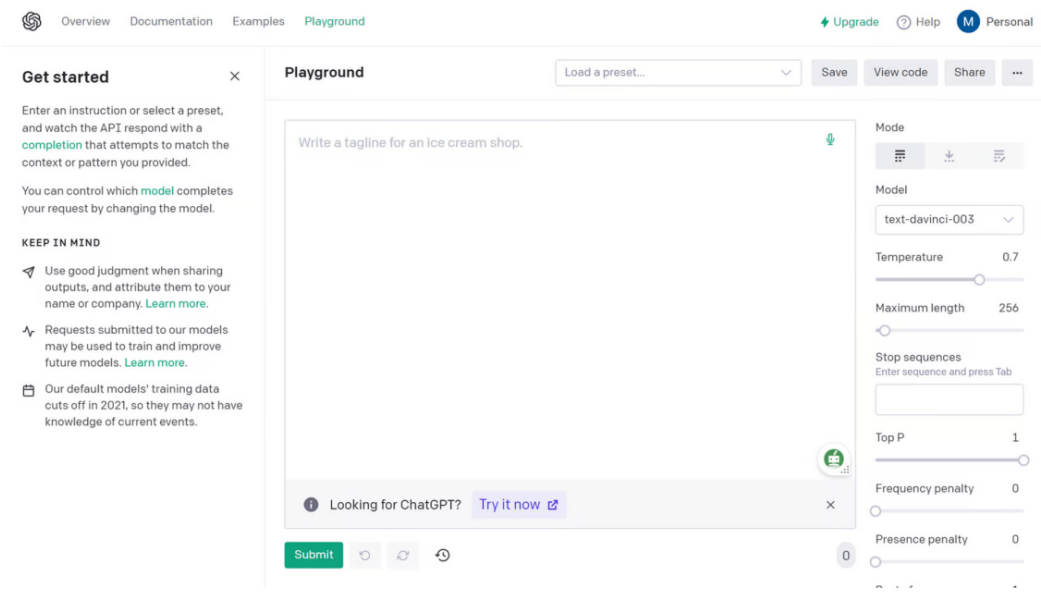

图 2-3 OpenAI Playground

语言模型可以基于一些说明了解和学习某些任务，而小样本提示正好可以赋能上下文学习能力。我们将在下一小节中介绍零样本提示和小样本提示等。提示词可以包含以下任意要素，具体例子如图 2-4 所示，该例子使用 ChatGPT 作为模型。

指令：想要模型执行的特定任务或指令。

上下文：包含外部信息或额外的上下文信息，引导语言模型更好地响应。

输入数据：用户输入的内容或问题。

输出指示：指定输出的类型或格式。

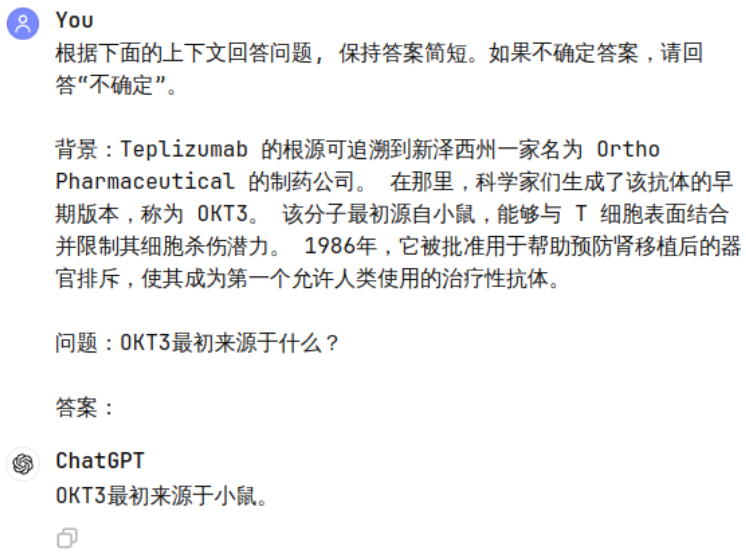

图 2-4 提示词使用示例图

2.2.2 典型提示词介绍

(1) 零样本提示

如今，经过大量数据训练并调整指令的大语言模型能够执行零样本任务。零样本提示就是不给出任何示例。

Wei 等人[25]研究发现，指令调整已被证明可以改善零样本学习。指令调整本质上是在通过指令描述的数据集上微调模型的概念。此外，RLHF（来自人类反馈的强化学习）已被采用以扩展指令调整，其中模型被调整以更好地适应人类偏好。这一最新发展推动了像 ChatGPT 这样的模型效果的提升。当零样本不起作用时，用户需要在提示中提供演示或示例，这就引出了少样本提示。

(2) 少样本提示

虽然大型语言模型展示了惊人的零样本能力，但在使用零样本设置时，它们在更复杂的任务上仍然表现不佳。少样本提示可以作为一种技术，以启用上下文学习，我们在

提示中提供演示以引导模型实现更好的性能。演示作为后续示例的条件，我们希望模型生成响应。Touvron[26]等人的在 2023 年的论文中表示，当模型规模足够大时，小样本提示特性开始出现标签空间和演示指定的输入文本的分布都很重要，无论标签是否对单个输入正确。使用的格式也对性能起着关键作用，即使只是使用随机标签，这也比没有标签好得多。所以使用一些少样本会使得模型表现的更好。

(3) 链式思考（CoT）提示

Wei 等人[27]于 2022 年的研究论文中引入的链式思考（CoT），如图 2-5 所示，提示通过中间推理步骤实现了复杂的推理能力。用户可以将其与少样本提示相结合，以获得更好的结果，以便在回答之前进行推理的更复杂的任务。

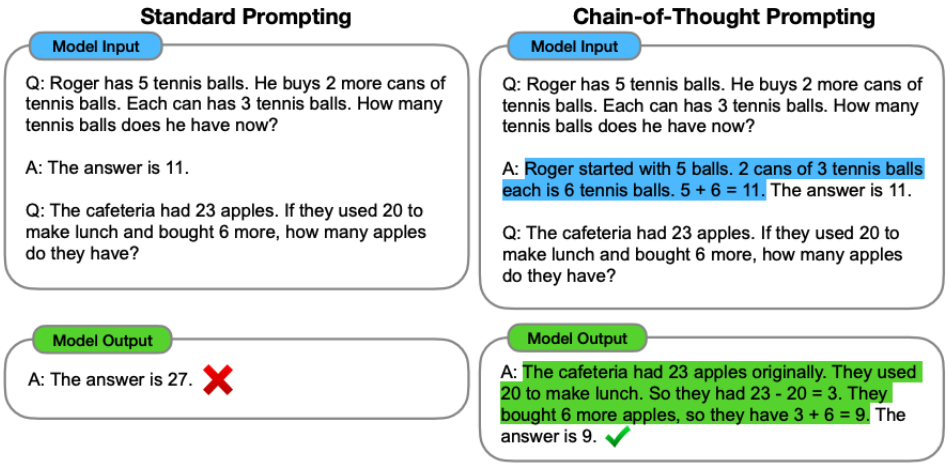

图 2-5 链式思考（CoT）[27]

(4) ReAct 框架

Yao[22]等人于 2022 年引入了一个提示框架，该框架让大语言模型以交错的方式生成推理轨迹和特定任务操作操作，如图 2-6 所示。生成推理轨迹使模型能够诱导、跟踪和更新操作计划，甚至处理异常情况。操作步骤允许与外部源（如知识库或环境）进行交互并且收集信息。ReAct 框架允许大语言模型与外部工具交互来获取额外信息，从而给出更可靠和实际的回应。

结果表明，ReAct 可以在语言和决策任务上的表现要高于几个最先进水准要求的的基线。ReAct 还提高了大语言模型的人类可解释性和可信度。总的来说，作者发现了将 ReAct 和链式思考（CoT）结合使用的最好方法是在推理过程同时使用内部知识和获取到的外部信息。

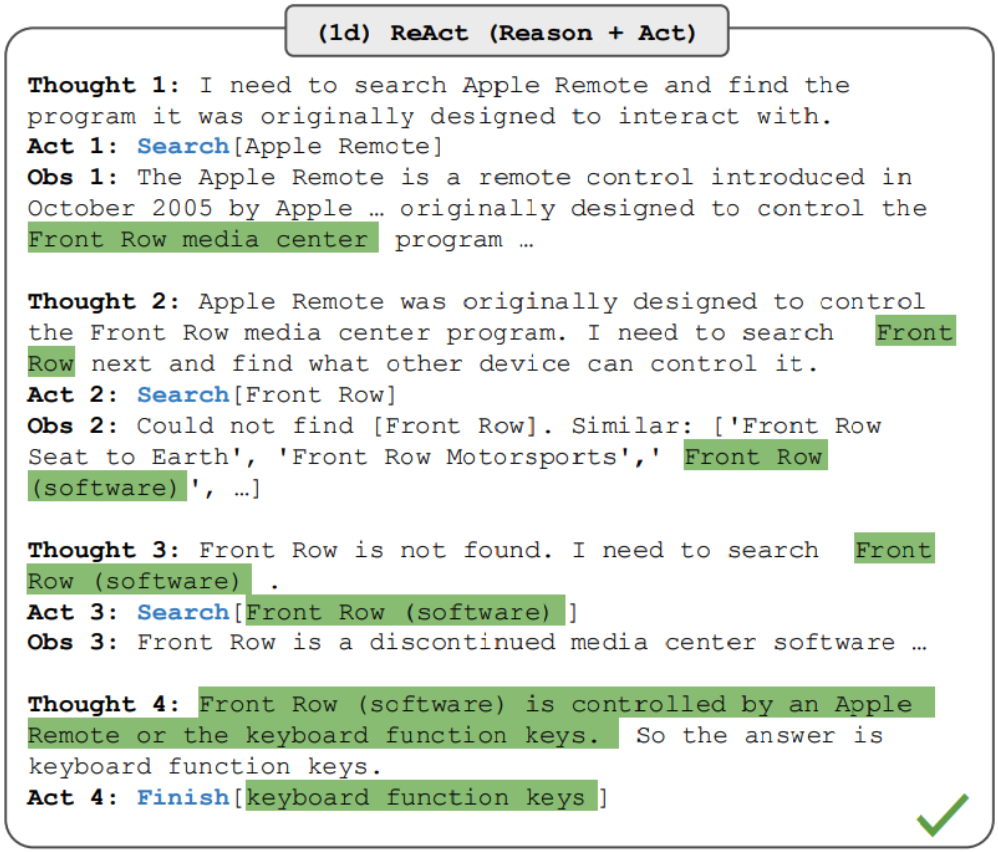

图 2-6 ReAct 框架[22]

2.3 智能体（Agent）

本文中的智能体，全称是 Large Language Model based Autonomous Agent，简称为 LLM Agent，或直接叫 Agent，指的是以大语言模型作为推理引擎的智能体。

智能体之所以受到关注，是因为借助大语言模型的推理能力，智能体可以在真正意义上，帮助人们解决一些现实的问题，不管是写代码，写文案，日程安排等，智能体都带来了无穷的可能性。智能体让大语言模型具备目标实现能力，并通过自我激励循环来实现这个目标，其由 4 个部分组成，分别是人设、记忆、规划和行动。

(1) 人设（Profile）。人设就是让大语言模型扮演的角色。一般都是某个领域的专家，这样才能给用户提供建议。比如，可以扮演律师、心理咨询师、资深程序员，或者其他角色。最简单的设置人设的方式，就是给大语言模型发送一条系统消息，例如，你是一名资深的心理咨询师，请帮助用户解决心理相关的问题。这样就为大语言模型设置了一

个人设。当然了，每个人设要求大语言模型具备相关的领域知识。

(2) 记忆（Memory）。对智能体来说，记忆的作用和人类的记忆是相似的。通过记忆，可以让智能体积累知识和自我进化。智能体的记忆分成两种，短期记忆和长期记忆。短期记忆，指的是当前大语言模型的上下文窗口中的内容。受限于上下文窗口的尺寸限制，短期记忆的内容一般比较少，只是最近与大语言模型的聊天记录。长期记忆，保存在外部储存中，通常是向量数据库中。在交互时，智能体可以提取与当前用户相关的长期记忆，作为生成当前内容时的辅助。

(3) 规划（Planning）。规划指的是，对于一个任务，智能体以什么样的方式来进行思考，找到完成任务的方式。规划的方式分成两种，无反馈和有反馈。无反馈的方式，指的是智能体根据预先设置的方式来执行任务，并不考虑每一步执行之后会带来的影响。在给大语言模型的提示中，就已经给出了具体的步骤。有反馈的方式，指的是智能体在每一步执行时，会考虑上一步执行了之后所造成的影响，由此来确定下一步的动作。有反馈的方式，适合于执行耗时较长，外部依赖多的复杂任务。

(4) 行动（Action）。行动模块把智能体的决策转换为具体的结果。这也是使用智能体的意义所在，对真实的世界产生影响。可以让智能体完成某些任务，或是输出一些内容。智能体在行动时，通常需要外部工具的支持。比如，如果你希望智能体帮你规划一天的日程安排，智能体需要使用外部工具来更新你的日程表。这些工具，可以是 API，数据库和知识库，甚至是其他的大语言模型。

2.3.1 智能体决策

智能体基于 PDCA 模型，如图 2-7 所示，进行规划、执行、评估和反思。

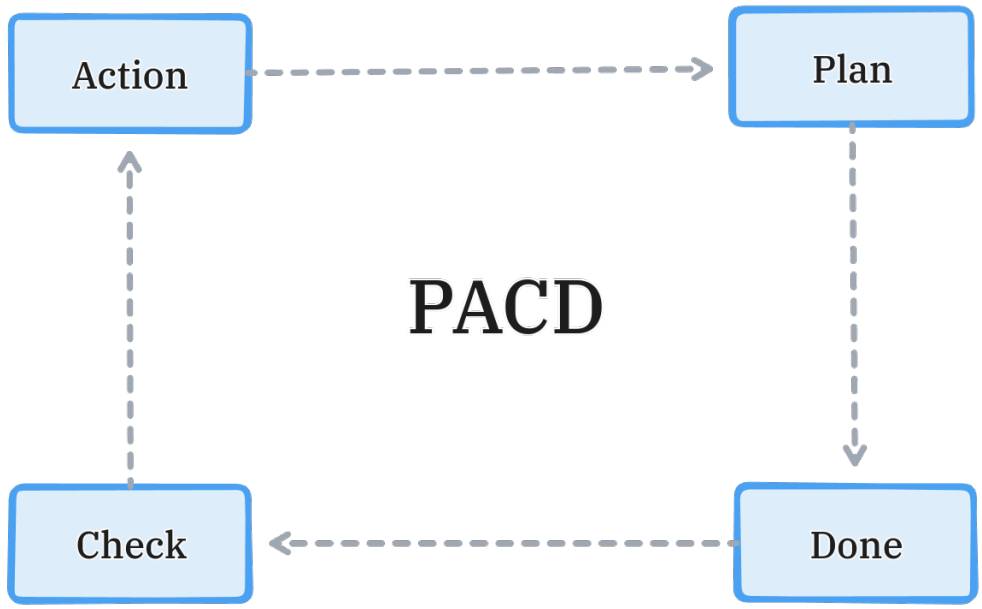

图 2-7 PACD 结构图

(1) 规划能力（Plan）- 分解任务。智能体大脑把大的任务拆解为更小的，可管理的子任务，这对有效的、可控的处理好大的复杂的任务效果很好。

(2) 执行能力（Done）- 使用工具。智能体能学习到在模型内部知识不够时（比如：在 pre-train 时不存在，且之后没法改变的模型 weights）去调用外部 API，比如：获取实时的信息、执行代码的能力、访问专有的信息知识库等等。这是一个典型的平台+工具的场景，我们要有生态意识，即我们构建平台以及一些必要的工具，然后大力吸引其他厂商提供更多的组件工具，形成生态。

(3) 评估能力（Check）- 确认执行结果。智能体要能在任务正常执行后判断产出物是否符合目标，在发生异常时要能对异常进行分类（危害等级），对异常进行定位（哪个子任务产生的错误），对异常进行原因分析（什么导致的异常）。这个能力是通用大语言模型不具备的，需要针对不同场景训练独有的小模型。

(4) 反思能力（Action）- 基于评估结果重新规划。智能体要能在产出物符合目标时及时结束任务，是整个流程最核心的部分；同时，进行归因分析总结导致成果的主要因素，另外，智能体要能在发生异常或产出物不符合目标时给出应对措施，并重新进行规划开启再循环过程。

由大语言模型赋能的智能体引发了人们对人工智能与人类工作的关系和未来发展的思考。它让我们思考人类如何与智能体合作，从而实现更高效的工作方式。而这种合

作方式也让我们反思人类自身的价值和特长所在。

2.3.2 智能体组件

智能体有三个关键组件，即规划、记忆和工具，其结构如图 2-8 所示。

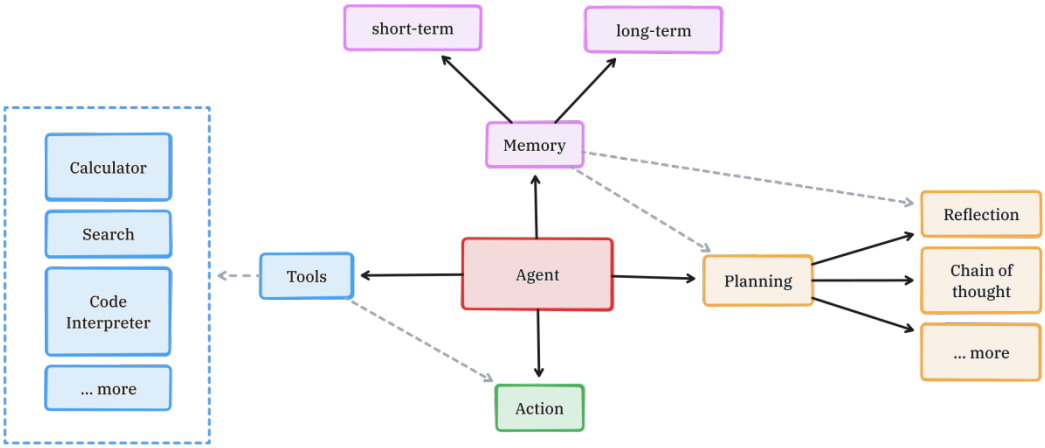

图 2-8 智能体组件图

(1) 规划（Planning）

规划组件分文子目标与分解（Subgoal and decomposition）和反思与完善（Reflection and refinement）。前者将大型任务分解为更小、更易于处理的子目标，从而实现对复杂任务的高效处理。后者可以对过去的行动进行自我批评和自我反思，自我反思可以让自主智能体改进过去的行动决策、纠正之前的错误来迭代改进，在可以试错的现实任务中非常有用。

在规划中，一项复杂的任务通常包括多个子步骤，智能体需要提前将任务分解，并进行规划。思维链（Chain of Thought, CoT）已然成为诱导模型推理的标准提示技术，可以增强解决复杂任务时的模型性能。ReAct 框架通过将动作空间扩展为任务相关的离散动作和语言空间的组合，在大语言模型中集成了推理和动作，其中动作使得大语言模型能够与环境交互（例如使用维基百科搜索 API），而语言空间可以让大语言模型以自然语言的方式生成推理轨迹。

(2) 记忆（Memory）

记忆组件分为短期记忆（Short-term memory）和长期记忆（Long-term memory）。

所有上下文学习（in-context learning）都是利用模型的短期记忆来学习，其非常短且影响范围有限，受到 Transformer 的上下文窗口长度的限制。

而长期记忆为代理提供了在长时间保留和回忆（无限）信息的能力，通常通过利用外部向量存储和快速检索来实现。获取长期记忆的方法最常见的方式是通过“语义搜索”，用一个 embedding 模型，将所有的记忆文本都转化为一个向量。而后续跟模型的交互信息也可以通过同样的 embedding 模型转化为向量，计算相似度来找到最相似的记忆文本。最后再将这些记忆文本拼接到 prompt 里，作为模型的输入。

(3) 工具使用（Tool use）

智能体可以学会调用外部应用程序接口（API）获取模型权重中缺失的额外信息（通常在预训练后很难更改），包括当前信息、代码执行能力、访问专有信息源等。使用复杂工具是人类高智力的体现，创造、修改和利用外部物体来完成超出身体和认知极限的事情，同样，为大语言模型配备外部工具也可以显著扩展模型功能。

MRKL（模块化推理、知识和语言），是一个神经符号架构的自主智能体，包含一组专家模块和一个用作路由器（router）的通用语言模型，以路由查询到最合适的专家模块。每个模块可以神经网络，也可以是符号模型，例如数学计算器、货币转换器、天气 API 等。

2.3.3 多智能体

智能体刚开始出现的时候，代表性的是 AutoGPT[28]， BabyAGI[29]（主张一个智能体来调用工具来解决复杂的任务），代表方法是 ReAct，总体上强调的是智能体使用工具的能力。后面出现了多智能体，侧重角色扮演（通过写提示词给智能体赋予一个角色，并规范一些行为），和各个智能体之间的协作，竞争等策略来完成一项复杂的任务，我觉得除了从底层支持和改进多智能体协作的能力外，还需要建设智能体自动化调度（广播机制，订阅发布者机制），多智能体通信协议，动态数量的智能体协作等，有了这些基础设施，就可以在原有的智能体基础上完成更高阶的任务。很直接的解释，多智能体像是一个团队，团队里面的智能体更像是不同的工种，通过不同的智能体之间的分工协作（有的做规划，有的执行，执行的时候可以调用工具，也可以利用大语言模型本身的代码，总结能力等），完成一些比较复杂的任务。

多智能体与单智能体（即一个 Agent 完成所有的任务）的区别：

(1) 多智能体的每个智能体只需要关注自己的立场，与自己相关的信息即可。不需要覆盖所有的历史信息，而单智能体，比如 AutoGPT，是需要记住所有的历史信息，意

味着单智能体在面临长历史的复杂任务时，对记忆容量（大语言模型支持的序列长度）要求比较高。多智能体在这方面有很明显的优势。

(2) 通过角色扮演的机制可以排除一些其他视角的观点，让大语言模型的表现更稳定，单智能体由于混杂了很多任务，无法做到这一点。

(3) 可拓展性更好，单智能体的拓展性依赖于 memory 的一些节省 token 的策略，对于更复杂的任务，每次输入给大语言模型的上下文会变长，会产生性能下降的风险（大语言模型处理长序列会存在丢失关键信息的情况等等）。多智能体分工协作则没有这个问题，因为每个智能体只完成特定的子任务，子任务一般不会造成很长的上下文。

(4) 可以多方案并行探索，然后选取最优的解决方案。单智能体则没有这种优势，或者实现起来会相对麻烦。

第 3 章　LangChain 开发框架

本章首先将详细介绍 LangChain 框架的内容及其中的核心组件，接着介绍 LangChain 框架的生态系统。第二章最后的内容讨论了 ReAct 框架的重要性，由于 LangChain 框架智能体组件的底层实现依托于该框架，所以在本章的最后将对 Agent 和 LangGraph 进行详细的介绍和描述。

3.1 LangChain 框架介绍

LangChain 框架是一个开源工具，充分利用了大语言模型的强大能力，以便开发各种下游应用，其结构如下图 3-1 所示。它的目标是为各种大语言模型应用提供通用接口，从而简化应用程序的开发流程。具体来说，LangChain 框架可以实现数据感知和环境互动，也就是说，它能够让语言模型与其他数据来源连接，并且允许语言模型与其所处的环境进行互动。LangChain 也提供了一套工具、组件和接口，可简化创建由大语言模型和聊天模型提供支持的应用程序的过程，其可以轻松管理与语言模型的交互，将多个组件链接在一起，并集成额外的资源，例如 API 和数据库。

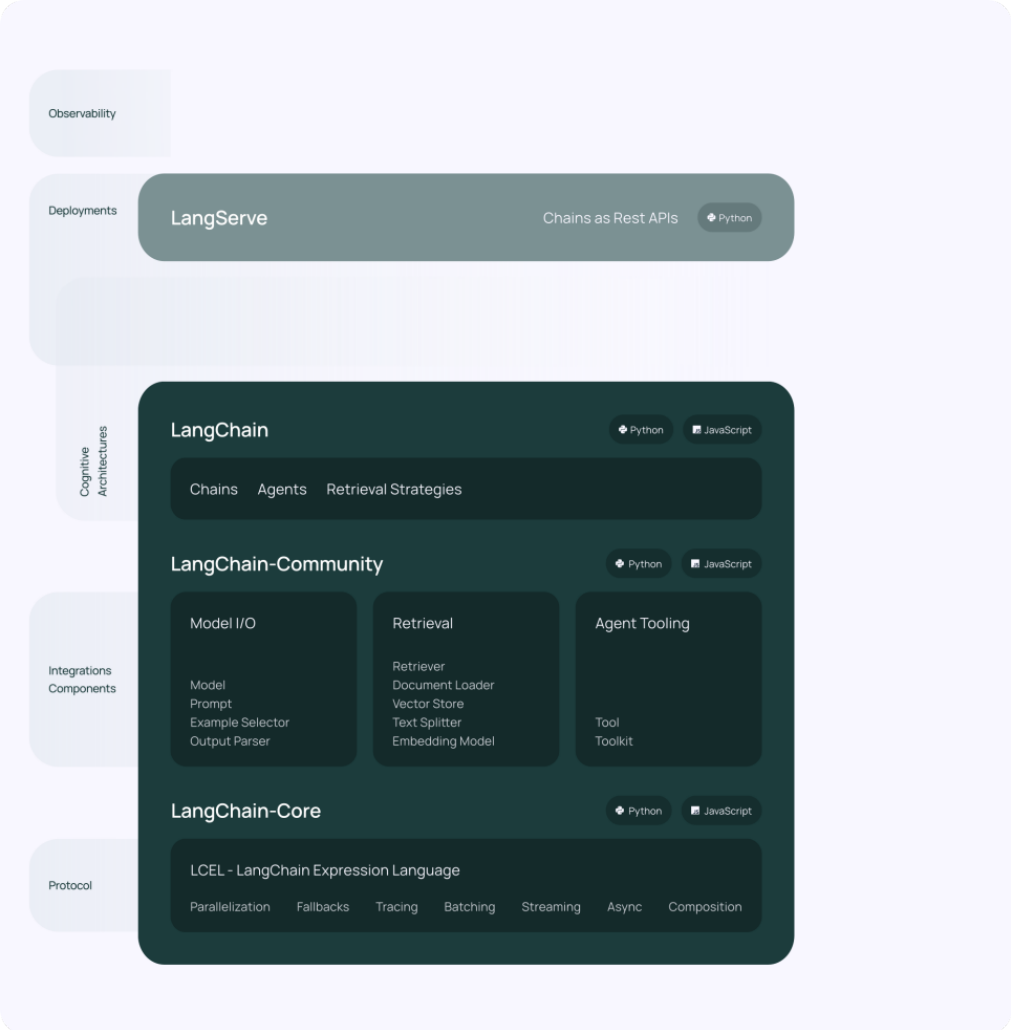

图 3-1 LangChain 框架结构[21]

在大语言模型技术领域的迅猛发展浪潮中，LangChain 作为一个不断进化的创新平台，持续推动着技术边界的拓展。2024 年 1 月 9 日，LangChain 正式发布了其稳定版本 v0.1.0，这一里程碑式的更新，为开发者带来了全面而强大的功能支持。其涵盖了模型的输入与输出处理、数据连接、链式操作、记忆机制、代理服务以及回调处理等关键组件，为 大语言模型 应用的开发和部署提供了坚实的基础。同时，LangChain 的持续优化和功能迭代，未来将带来更多创新特性和性能提升。

3.2 LangChain 表达式语言

LangChain 表达式语言（LangChain Expression Language，LCEL） 是 LangChain 框架的重要补充，旨在提高文本处理任务的效率和灵活性。LCEL 允许用户采用声明式方法来组合链，便于进行流处理、批处理和异步任务。其模块化架构还允许轻松定制和修

改链组件。LCEL 的优势是它使用户更容易个性化链的不同部分。链的声明性和模块化特性允许轻松地交换组件。除此之外，它让提示变得更加明显，可以轻松地修改以适应特定的用例。在 LangChain 中，提示只是默认值，但是可以被生产应用程序进行更改。

3.3 核心组件

(1) Model I/O

LangChain 在该部分提供了一些抽象类，可以使用它们更快捷方便的构建我们的应用程序，同时还有方便在代码中使用的输入处理和输出处理功能。Model I/O 组件结构如图 3-2 所示。

Model I/O 部分分为以下三个部分：

1) 提示：模板化、动态选择和管理模型输入，对大语言模型进行编程的新方法是通过提示词。提示指的是对模型的输入，这种输入通常是由多个组件构成的。LangChain 提供了与之相关的类和函数，使构建和处理提示信息变得容易。
2) 语言模型：通过通用接口调用语言模型，如 ChatGPT 模型或百度千帆大语言模型。
3) 输出分析器：从模型输出中提取需要的信息，可将输出修改为 Json 格式或字符串格式。

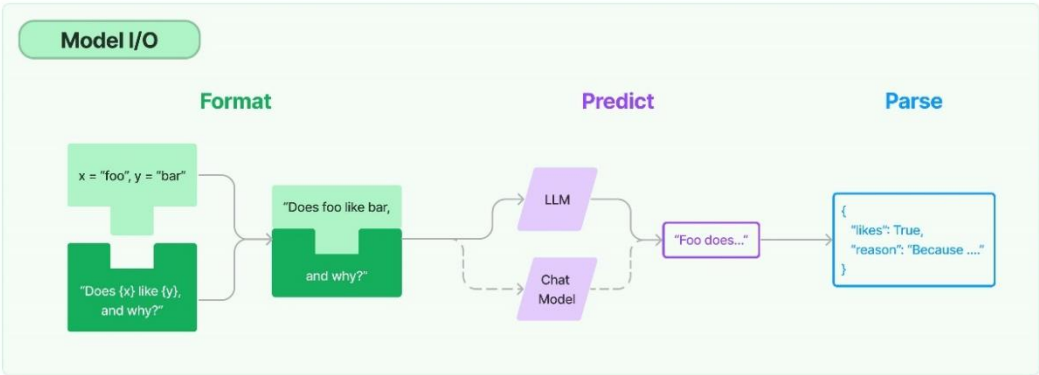

图 3-2 Model I/O[21]

(2) 检索（Retrieval）

许多大语言模型需要特定于用户的数据，这些数据不属于模型训练集的一部分。实

现这一目标的主要方法是通过检索增强生成（RAG）。在此过程中，检索工具将外部数据加载后，在执行生成步骤时将其传递给大语言模型，本小节将按照图 3-3 中的流程介绍在检索过程中所需要的工具。

检索过程首先需要使用文档加载器加载原始数据。LangChain 提供 100 多种不同的文档加载器来加载不同来源的文档，并与该领域的其他主要数据提供商进行集成。 其提供了从所有类型的位置（私有存储库、公共网站）加载所有类型文档（HTML、PDF、代码）的集成。

当原始文本数据加载完成之后需要进行文本分割，其关键部分是获取文档的相关内容。这涉及几个转换步骤来准备文档以供检索。这里的主要任务之一是将大文档分割成为更小的块。LangChain 提供了多种转换算法来执行此操作，以及针对特定文档类型（代码、Markdown 等）优化的逻辑。

检索工具的另一个关键部分是为文档创建嵌入模型（Embedding）。嵌入模型将捕获文本的语义，使用户能够快速有效地找到文本的其他相似部分。 针对嵌入模型，LangChain 提供与超过 25 种不同嵌入提供商和方法的集成，让用户可以选择最适合需求的一种，LangChain 也提供了标准的接口，让用户可以轻松地在模型之间进行切换。

矢量存储（Vector Store）随着嵌入式模型的兴起，检索过程需要数据库来支持这些嵌入的高效存储和搜索。 LangChain 提供与 50 多种不同矢量存储的集成，从开源本地矢量存储到云托管专有矢量存储，让用户可以选择适合需求的一种。 LangChain 公开了一个标准接口，让用户可以在向量存储之间进行交换。 LangChain 支持易于上手的基本方法——即简单语义搜索。然而，我们还在此基础上添加了一系列算法以提高性能。这些包括：

1) 父文档检索器：该检索器允许为每个父文档创建多个嵌入，从而允许您查找较小的块但返回更大的上下文。
2) 自查询检索器：用户的问题通常包含对某些内容的引用，这些内容不仅是语义的，而且表达了一些可能为元数据的逻辑。 自查询允许您从查询中存在的其他元数据过滤器解析出查询的语义部分。
3) 集成检索器：有时用户可能希望从多个不同的源检索文档，或使用多种不同的算法，集成检索器提供了这些功能。

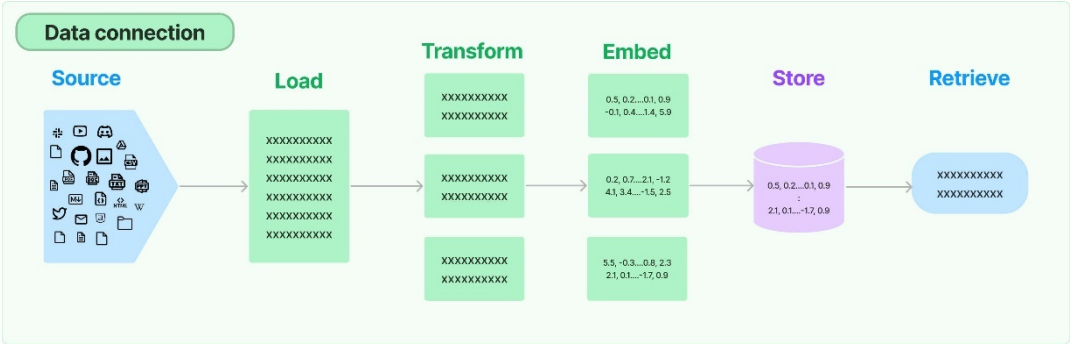

图 3-3 Retrieval[21]

(3) 索引（Indexing）

LangChain 的索引功能（Indexing）用于将用户的数据从任何来源同步到向量存储中，帮助用户避免将重复的内容写入矢量存储库、避免重写未更改的内容和避免在未更改的内容上重新计算嵌入。该工具可以节省用户的时间和金钱，并改善矢量搜索结果。

(4) 智能体（Agent）

智能体的核心思想是使用大语言模型来选择要采取的一系列操作。在链中，一系列操作被硬编码在代码中。在智能体中，语言模型被用作推理引擎来确定要采取哪些操作以及按什么顺序。

(5) 链（Chain）

链是指调用序列——无论是大语言模型、工具还是数据预处理步骤。主要支持的方法是使用 LCEL。LCEL 非常适合构建链条，但使用现成的链条也很好。LangChain 支持的现成链有两种：使用 LCEL 构建的链。在这种情况下，LangChain 提供了更高级的构造方法。然而，幕后所做的一切都是用 LCEL 构建一条链。这些链在底层不使用 LCEL，而是独立的类。LangChain 团队正在致力于创建创建所有链的 LCEL 版本。以这种方式构造的链很好，因为如果用户想修改链的内部结构，可以简单地修改 LCEL。这些链本身支持开箱即用的流式传输、异步和批处理。

3.4 LangChain 生态

(1) LangSmith

一个智能助手的性能并非一成不变。随着数据量的增加和用户需求的变化，模型需要不断地进行训练和优化。这就是 LangSmith 发挥作用的地方。作为框架的模型训练和

优化工具，LangSmith 为开发者提供了丰富的算法和技巧。它支持多种模型训练方式，包括有监督学习、无监督学习等，以帮助开发者不断提升模型性能。这意味着开发者可以利用 LangSmith 来优化他们的智能助手，使其更加准确、可靠，并更好地满足用户需求。

(2) LangServe

LangServe 为开发者提供了稳定、可靠的服务端支持。作为框架的服务端组件，LangServe 负责提供高效且稳定的在线服务。它支持多种部署方式，包括云服务器、容器化部署等，以满足不同场景下的需求。通过 LangServe，开发者可以将智能助手快速部署到线上，为用户提供实时、准确的回答和交互。这不仅提高了用户体验，也增强了智能助手的可用性和可靠性。

(3) LangGraph

LangGraph 是一个用于使用大语言模型构建有状态多角色应用程序的库。受 Pregel[30]和 Apache Beam[31] 的启发，LangGraph 可让用户使用普通 Python 函数在循环计算步骤中协调和检查多个链（或角色）。LangGraph 支持使用 LangChain 表达式语言来快速的构建有向无环图（DAG）来实现有状态多角色应用的程序。

循环对于代理行为非常重要，在代理行为中，用户可以循环调用大语言模型，询问它在下一步要采取什么行动，通过这种方式我们可以观测到整个应用程序的思考过程，体现其智能的特点。

3.5 Agent 与 LangGraph

上文介绍了智能体（Agent)和 LangGraph 的基本概念，本文所构建的智能助手程序主要由智能体和 LangGraph 构成。本小节将详细介绍两者的联系，为下一章节构建智能助手打下基础。

3.5.1 LangChain 智能体（Agent）原理

LangChain 框架中的智能体的实现依赖于提示工程中的提示框架，在 LangChain 框架中最常见的智能体为 ReAct（推理&行动）范式的智能体，该智能体让大语言模型增加了工具的使用能力。通过在提示中注入可用工具信息，大语言模型将自动调用工具并

能并能自动调用工具以获得对应的结果。除此之外，使用 ReAct 提示框架的大语言模型也了增加“循环”能力。智能体的运行通常需要多次推理（Reason）——行动（Act）的反复与循环，直到完成任务。

LangChain 可以构建单一智能体完成一种任务，这也带来了一些问题，如果把单一智能体想象成图中的节点，组件之间的联系想象成图中的边，那么这个智能体就是一个有向无环图(DAG)。即在一次运行中，一个调用节点无法重复进入。LangGraph 的出现弥补了上述的缺点。

3.5.2 LangGraph 的驱动力

LangGraph 并非一个独立于 LangChain 的新框架，它是基于 LangChain 之上构建的一个扩展库, 可以与 LangChain 现有的链、LangChain 表达式语言等无缝协作。LangGraph 能够协调多个链、智能体、工具等共同协作来完成输入任务，支持大语言模型调用“循环”以及智能体过程的更精细化的控制。

LangGraph 的实现方式是把原先基于智能体执行结构（AgentExecutor）的黑盒调用过程用一种新的形式来构建：状态图（StateGraph）。把基于大语言模型的任务（比如 RAG、代码生成等）细节用图的形式进行精确的定义，即定义图的节点与边，最后基于这个图来编译生成应用；在任务运行过程中，维持一个中央状态对象(state)，会根据节点的跳转不断更新，状态包含的属性可自行定义。

在 LangGraph 中定义了如下集中状态：

(1) StateGraph。代表整个状态图的基础类。

(2) Nodes（节点）。在有了图之后，可以向图中添加节点，节点通常是一个可调用的函数、一个可运行的 Chain 或者 Agent。有一个特殊的节点叫 END，进入这个节点，代表运行结束。在图 3-4 中，推理函数调用、调用检索器、生成响应内容、问题重写等都是其中的任务节点。

(3) Edges（边）。有了节点后，需要向图中添加边，边代表从上一个节点跳转到下一个节点的关系。目前有三种类型的边：

(4) Starting Edge（一种特殊的边）。用来定义任务运行的开始节点，所以它没有上一个节点。

(5) Normal Edge（普通边）。代表上一个节点运行完成后立即进入下一个节点。比

如在调用 Tools 后获得结果后，立刻进入大语言模型推理节点。

(6) Conditional Edge（条件边）。代表上一个节点运行完成后，需要根据条件跳转到某个节点，因此这种边不仅需要上游节点、下游节点，还需要一个条件函数，根据条件函数的返回来决定下游节点。

第 4 章 基于 Langchain 的个人智能助手设计与实现

第二章介绍了个人智能助手的相关技术，第三章介绍了 LangChain 框架和 LangGraph 扩展库。这一章将上述知识结合起来，完成构建基于大语言模型的个人智能助手的任务。

该智能助手系统的实现将分为交互界面（前端）与对话处理（后端）两个部分。交互界面部分将使用 Streamlit 框架进行构建，对话处理部分将使用 LangChain 进行实现。本文在大语言模型聊天功能的基础上，扩展至与网络搜索功能、文本交互功能和邮件处理系统等多方面，以此提供更加丰富的服务支持，助力人们高效完成各类工作任务。

4.1 实验环境

本系统使用 Python 语言进行实现，使用 Poetry 包管理工具进行项目管理，所使用 LangChain 版本为 0.1.0 稳定发行版，Python 版本为 3.11，开发环境系统为 Linux。

4.2 应用场景描述

在本文第一章的第三小节中，使用了三个应用场景的例子来展示本文设计的智能助手的功能，本小节将分别对这三个应用场景做出详细的解释，并针对这些应用场景对其实现逻辑进行分析设计。

(1) 场景一：邮件分发任务

场景描述：当企业管理者计划在周末组织团队建设活动时，他需要先通过网络搜索了解次日的天气状况，随后撰写相应的电子邮件分发给员工。

本文实现的个人智能助手将简化上述流程，将这些步骤环节通过一句简单的对话或指令即可完成。为了应对此应用场景，本文使用一种具有管理者的结构，该结构如图 4-1 所示，该结构首先创建一个智能体组，并定义一个大语言模型接口充当管理者的角色，使用提示词技术让该管理者决定下一个节点的走向，并判断整个任务流程是否结束，然后分别定义网络搜索者、文件管理者和文件处理者的角色，来处理具体的任务。

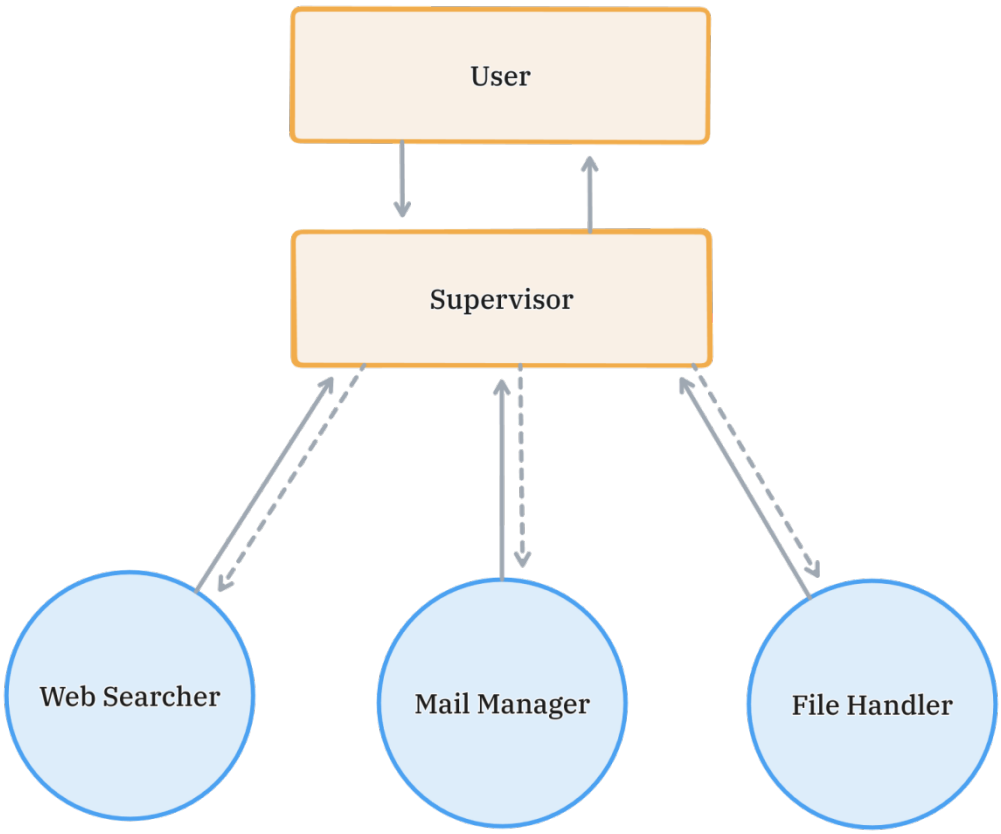

图 4-1 管理者结构

在该场景中，用户需要先与管理者交互，给予管理者任务，随后管理者会分析用户的任务，首先使用网络搜索者查询今天的天气情况，查询完成之后会再次返回给管理者，管理者会再次决定，由邮件管理者将天气的信息通过邮件发送给目标。

(2) 场景二：文件处理任务

场景描述：员工在工作过程中经常会遇到大量的邮件，需要对这些邮件进行整理，提取电子邮件中的重要信息，并将之整理。

针对上述场景，本文介绍的个人智能助手将帮助完成读取文件的任务，并将其写入到文本文件中，让用户可以更加轻松的查看这些邮件的内容信息。

该应用场景与场景一中使用相同的结构来完成任务，结构如上图 4-1 所示，但在该结构中管理者会使用不同的智能体成员来完成该场景中的任务，首先管理者会让邮件管理者查询收到的邮件，然后再通过文件处理者写入到本地文件中。

(3) 场景三：问题研究任务

场景描述：当员工需要针对上级领导提出的某项问题展开深入研究时。

针对上述场景，个人助手，将通过网络搜索收集资料，对用户给出的问题进行分析调研，最终生成一份研究报告提供给用户查看。受最近的 STORM[32]的启发，本文构建一个能够自行进行研究的助手，通过使用 LangGraph，利用具有专业技能的多个代理来显着提高研究过程的深度和质量。本节展示了智能体团队如何协同工作，对给定主题进行从规划到发布的研究。最终，平均一次运行会生成 5-6 页的研究报告，采用以下多种格式：PDF、Docx 和 Markdown。

研究团队由 7 名智能体成员组成，它们分别扮演不同的角色，结构和工作流程如图 4-2 所示，具体角色如下：

1) 主编：监督研究过程并管理团队。这是使用 LangGraph 协调其他智能体的“主”智能体。
2) 研究员：进行网络搜索等收集信息，对给定主题进行深入研究。
3) 编辑：负责规划研究大纲和结构。
4) 审阅者：根据用户给定的标准验证研究结果的正确性。
5) 修订者：根据审稿人的反馈修改研究结果。
6) 作家：负责编译和撰写最终报告。
7) 发布者：负责以各种格式发布最终报告。

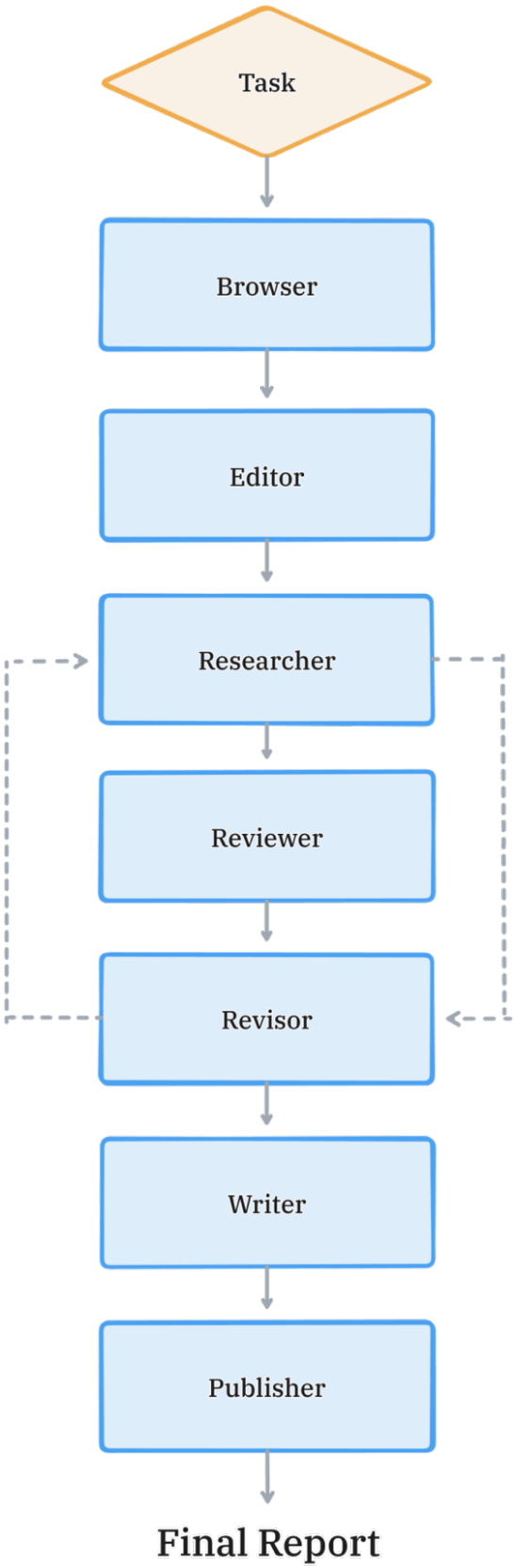

图 4-2 问题研究场景智能体团队结构

4.3 智能助手功能实现

4.3.1 基本功能

本节将使详细展示个人智能助手的基本功能，其拥有普通聊天机器人所具备的所有基本功能。当用户进行多轮对话时，用户和助手的对话将被保存下来，用户可以随时进行查看，如下图 4-3 所示，在进行多轮对话之后，该助手还能够记住之前的聊天信息。

图 4-3 会话历史消息展示

该智能助手可以展示其思考过程，用户可以清楚的查看该智能助手是否使用了额外的智能体功能，以及是否联合使用了多个智能体，如下图 4-4 所示。该机器人通过谷歌搜索得到了 2024 年 5 月 12 日武汉的天气情况，并对查询内容做出总结。

图 4-4 网络查询功能 1

当用户对比较新的事情进行询问时，该智能助手也能正确回答，如下图 4-5 所示，图中显示再对 2024 年奥运会进行询问时，它能够正确回答用户提出的问题，这是一般大语言模型所不具备的。

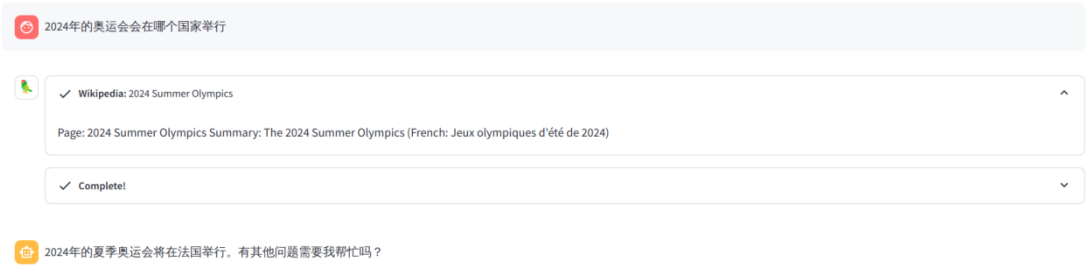

图 4-5 网络查询功能 2

普通大语言模型对计算能力一般较弱，该智能助手加入了计算模块可以更准确的计算出答案，如下图 4-6 所示，该智能助手调用了计算模块正确计算了乘法的结果。

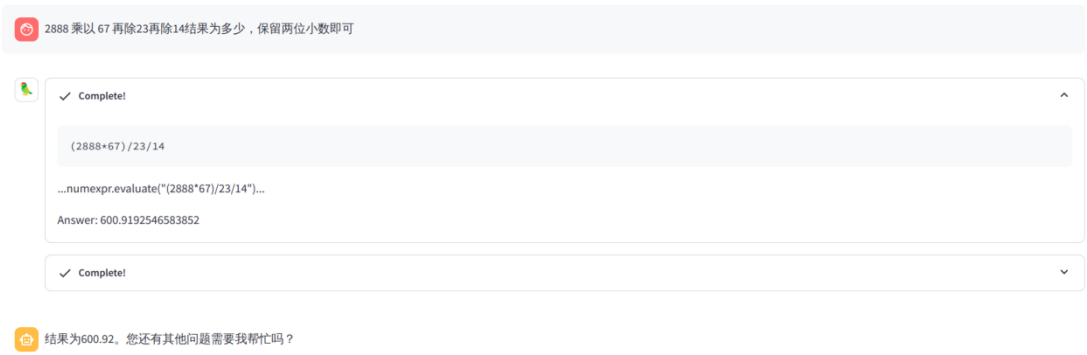

图 4-6 数值计算功能

该智能助手具备文本检索功能，上传文件后智能助手即可回答文件中的主要内容，如下图图 4-7 所示，该智能助手准确的概括了上传文章中的简要信息。

图 4-7 文本检索功能

4.3.2 多智能体协作功能

(1) 场景一：邮件分发任务的个人智能助手功能实现

在该场景中，用户首先需要通过交互界面中输入任务并提交给智能助手，之后由上文中提到的管理者结构中的管理者对任务进行处理。该场景中的任务流程如图 4-8 所示。

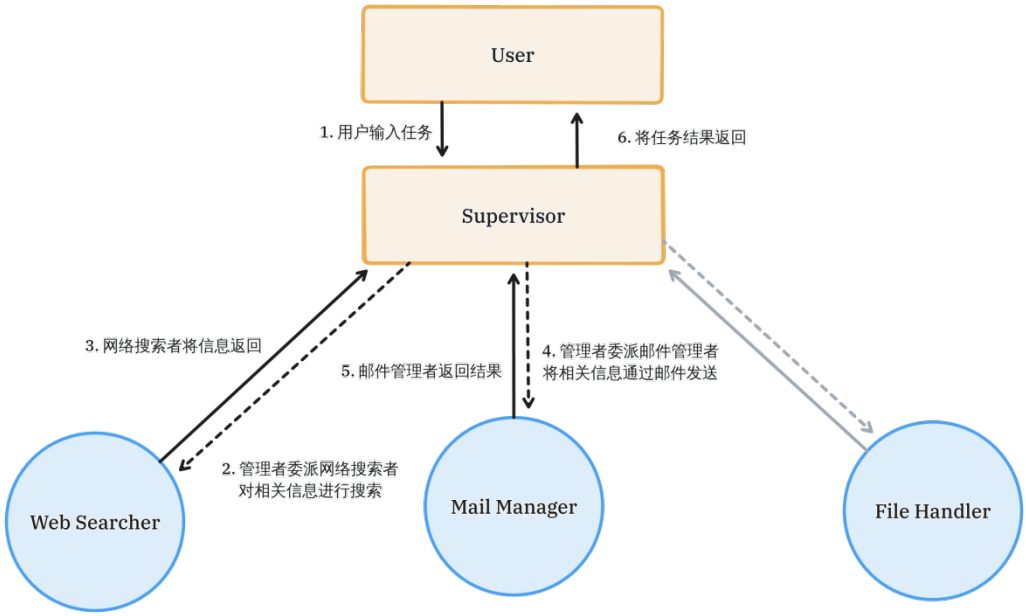

图 4-8 邮件分发场景任务流程

个人智能助手也会通过可视化界面展示该场景中的任务流程，如图 4-9 所示。最终天气的信息会通过邮件发送，如图 4-10 所示。

图 4-9 邮件分发场景可视化展示图

图 4-10 邮件分发场景结果图

(2) 场景二：文件处理任务的智能助手实现

该场景的任务流程与场景一中的任务流程基本一致，不同的是在该场景的任务中管理者首先会委派邮件管理者获取邮件信息，然后再通过文件处理者将信息写入文件中。该场景中的任务流程如图 4-11 所示。

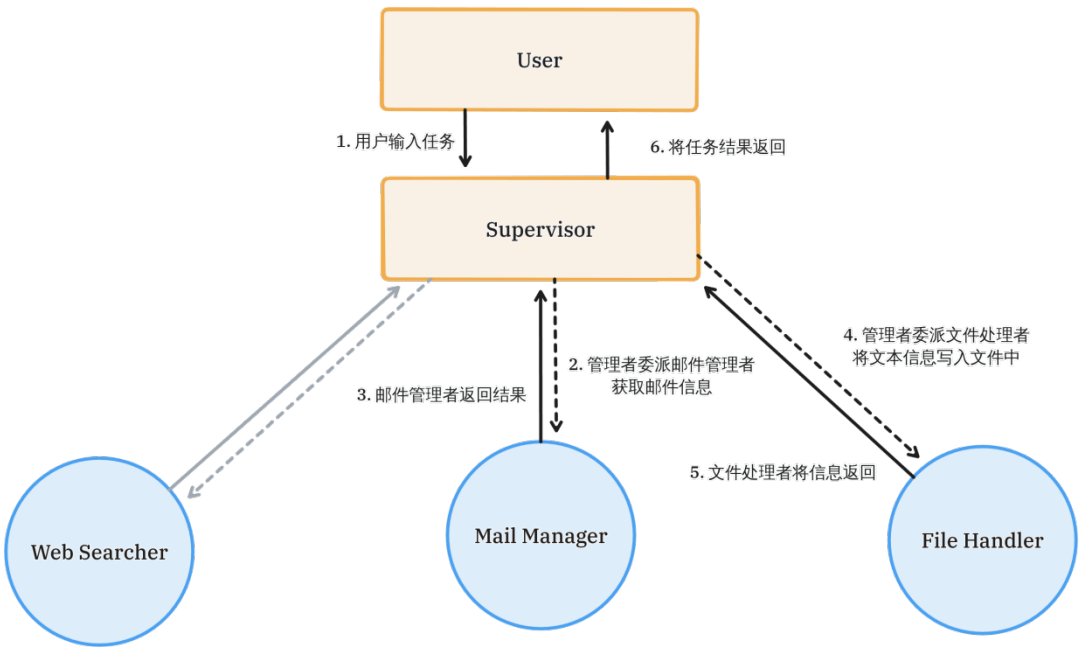

图 4-11 文件处理场景任务流程

个人智能助手在该场景中也会通过可视化界面展示该场景中的任务流程，如图 4-12 所示。邮件中的信息会被写进文本文件中，如图 4-13 所示。

图 4-12 文件处理场景可视化展示图

图 4-13 文件处理场景结果图

(3) 场景三：问题研究任务的智能助手实现

在问题研究场景的任务中，需要使用更多个智能体进行角色扮演，组成一个团队，各司其职，最终共同完成对某个话题的研究。话题运行时如图 4-14 所示，在该任务中，用户不仅可以输入需要研究的问题，也可以提出自己的要求，例如，需要使用什么样的格式编写文档，编写文档用什么语言等等。最终得到的报告前两页结果如图 4-15 所示。

图 4-14 问题研究场景可视化展示图

图 4-15 问题研究场景结果图

第 5 章 总结与展望

5.1 总结

ChatGPT 等大语言模型的出现给个人智能助手领域带来了颠覆性的改变，LangChain 的诞生给开发者对于大语言模型应用的开发提供了更多的可能性。针对以往传统智能助手在功能方面存在的局限以及无法即时获取新鲜资讯的弊端，通过 LangChain 实现的个人智能助手具有更加实用、即时、综合的功能给用户提供了便利，帮助用户解决工作中繁琐的事务，让用户更加注重重要的工作。

本文对大语言模型提示词工程中的智能体部分和 LangChain 框架进行了深入的研究，探讨了它们在提升大语言模型性能和效率方面的重要作用。在智能体部分，我们详细介绍了其在大语言模型领域的基本原理和单一智能体的局限性，并通过 LangGraph 库构造了一种状态图来解决单一智能体只具有一种状态的问题。最终实现的个人智能助手能够切实帮助用户解决繁琐的任务。

5.2 展望

本文实现的个人智能助手也存在依赖官方应用程序接口，稳定性不足等缺点，未来也将会更深入地探索多智能体的可能性，让个人智能助手更加智能。

参考文献

[1] Aron J. How innovative is Apple's new voice assistant, Siri?[J]. 2011.

[2] Kamble A, Mulani A O. Google assistant based device control[J]. Int. J. of Aquatic Science, 2022, 13(1): 550-555.

[3] Tulshan A S, Dhage S N. Survey on virtual assistant: Google assistant, siri, cortana, alexa[C]//Advances in Signal Processing and Intelligent Recognition Systems: 4th International Symposium SIRS 2018, Bangalore, India, September 19–22, 2018, Revised Selected Papers 4. Springer Singapore, 2019: 190-201.

[4] Van Dijck J. From shoebox to performative agent: the computer as personal memory machine[J]. New Media & Society, 2005, 7(3): 311-332.

[5] Lowerre B P, Reddy B R. Harpy, a connected speech recognition system[J]. The Journal of the Acoustical Society of America, 1976, 59(S1): S97-S97.

[6] Norris J R. Markov chains[M]. Cambridge university press, 1998.

[7] Fassbender E, Mamtora J. A workflow for managing information for research using the iPad, Sente and Dragon Dictate: a collaboration between an academic and a research librarian[J]. The Australian Library J ournal, 2013, 62(1): 53-60.

[8] Morton J, Marcus S, Frankish C. Perceptual centers (P-centers)[J]. Psychological review, 1976, 83(5): 405.

[9] Lai J, Vergo J. MedSpeak: Report creation with continuous speech recognition[C]//Proceedings of the ACM SIGCHI Conference on Human factors in computing systems. 1997: 431-438.

[10] Hoy M B. Alexa, Siri, Cortana, and more: an introduction to voice assistants[J]. Medical reference services quarterly, 2018, 37(1): 81-88.

[11] Wu T, He S, Liu J, et al. A brief overview of ChatGPT: The history, status quo and potential future development[J]. IEEE/CAA Journal of Automatica Sinica, 2023, 10(5): 1122-1136.

[12] Perez I, Dedden F, Goodloe A. Copilot 3[R]. 2020.

[13] Poon A Y K, Lau C M Y, Chu D H W. Impact of the fine-tuning medium-of-instruction policy on learning: Some preliminary findings[J]. Literacy Information and Computer Education Journal, 2013, 4(1): 946-954.

[14] Sun Z, Shen S, Cao S, et al. Aligning large multimodal models with factually augmented rlhf[J].

arXiv preprint arXiv:2309.14525, 2023.

[15] Achiam J, Adler S, Agarwal S, et al. Gpt-4 technical report[J]. arXiv preprint arXiv:2303.08774, 2023.

[16] Rudolph J, Tan S, Tan S. War of the chatbots: Bard, Bing Chat, ChatGPT, Ernie and beyond. The new AI gold rush and its impact on higher education[J]. Journal of Applied Learning and Teaching, 2023, 6(1).

[17] Aydın Ö. Google Bard generated literature review: metaverse[J]. Journal of AI, 2023, 7(1): 1-14.

[18] Team G, Anil R, Borgeaud S, et al. Gemini: a family of highly capable multimodal models[J]. arXiv preprint arXiv:2312.11805, 2023.

[19] Zeng W, Ren X, Su T, et al. PanGu-$\alpha $: Large-scale Autoregressive Pretrained Chinese Language Models with Auto-parallel Computation[J]. arXiv preprint arXiv:2104.12369, 2021.

[20] Zhang Q, Wang H, Liu C. MILM hybrid identification method of fractional order neural-fuzzy Hammerstein model[J]. Nonlinear Dynamics, 2022, 108(3): 2337-2351.

[21] LangChain.LangChain Document[DB/OL].https://www.langchain.com, 2024.

[22] Yao S, Zhao J, Yu D, et al. React: Synergizing reasoning and acting in language models[J]. arXiv preprint arXiv:2210.03629, 2022.

[23] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[24] Friederich S. Fine-tuning[J]. The Stanford encyclopedia of philosophy, 2017.

[25] Wei J, Bosma M, Zhao V Y, et al. Finetuned language models are zero-shot learners[J]. arXiv preprint arXiv:2109.01652, 2021.

[26] Touvron H, Lavril T, Izacard G, et al. Llama: Open and efficient foundation language models[J]. arXiv preprint arXiv:2302.13971, 2023.

[27] Wei J, Wang X, Schuurmans D, et al. Chain-of-thought prompting elicits reasoning in large language models[J]. Advances in neural information processing systems, 2022, 35: 24824-24837.

[28] Yang H, Yue S, He Y. Auto-gpt for online decision making: Benchmarks and additional opinions[J]. arXiv preprint arXiv:2306.02224, 2023.

[29] Talebirad Y, Nadiri A. Multi-agent collaboration: Harnessing the power of intelligent llm agents[J]. arXiv preprint arXiv:2306.03314, 2023.

[30] Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large-scale graph

processing[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. 2010: 135-146.

[31] Spæren T. Performance analysis and improvements for Apache Beam[D]. , 2021.

[32] Shao Y, Jiang Y, Kanell T A, et al. Assisting in Writing Wikipedia-like Articles From Scratch with Large Language Models[J]. arXiv preprint arXiv:2402.14207, 20

[33] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

[34] Han Z, Gao C, Liu J, et al. Parameter-efficient fine-tuning for large models: A comprehensive survey[J]. arXiv preprint arXiv:2403.14608, 2024.

致谢

感谢所有帮助过我的人，你们的支持和鼓励是我前进的动力。我将带着你们的期望和祝福，继续前行，追求更高的人生目标。