上海交通大学硕士学位论文

上海交通大学学位论文 LaTeX 模板示例文档

姓　　名：蔡卓悦

学　　号：122037910055

导　　师：吴刚教授

院　　系：软件学院

学 科/专 业：软件工程

申 请 学 位：专业学位硕士

2024 年 11 月 10 日

A Dissertation Submitted to

Shanghai Jiao Tong University for the Degree of Master

A SAMPLE DOCUMENT FOR LaTeX-BASED SJTU THESIS TEMPLATE

Author: Cai Zhuoyue

Supervisor: Prof. Wu Gang

School of Software

Shanghai Jiao Tong University

Shanghai, P.R. China

November 10^{th}, 2024

上海交通大学

学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全知晓本声明的法律后果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

上海交通大学

学位论文使用授权书

本人同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。

本学位论文属于：

□**公开论文**

□**内部论文**，保密□ 1 年/□ 2 年/□ 3 年，过保密期后适用本授权书。

□**秘密论文**，保密____ 年（不超过 10 年），过保密期后适用本授权书。

□**机密论文**，保密____ 年（不超过 20 年），过保密期后适用本授权书。

（请在以上方框内选择打“✓”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

摘 要

中文摘要应该将学位论文的内容要点简短明了地表达出来，应该包含论文中的基本信息，体现科研工作的核心思想。摘要内容应涉及本项科研工作的目的和意义、研究方法、研究成果、结论及意义。注意突出学位论文中具有创新性的成果和新见解的部分。摘要中不宜使用公式、化学结构式、图表和非公知公用的符号和术语，不标注引用文献编号。硕士学位论文中文摘要字数为 500 字左右，博士学位论文中文摘要字数为 800 字左右。英文摘要内容应与中文摘要内容一致。

摘要页的下方注明本文的关键词（4 ~ 6 个）。

关键词：上海交大，饮水思源，爱国荣校

Abstract

Shanghai Jiao Tong University (SJTU) is a key university in China. SJTU was founded in 1896. It is one of the oldest universities in China. The University has nurtured large numbers of outstanding figures include JIANG Zemin, DING Guangen, QIAN Xuesen, Wu Wenjun, WANG An, etc.

SJTU has beautiful campuses, Bao Zhaolong Library, Various laboratories. It has been actively involved in international academic exchange programs. It is the center of CERNet in east China region, through computer networks, SJTU has faster and closer connection with the world.

Key words: SJTU, master thesis, XeTeX/LaTeX template

目 录

插 图

表 格

算 法

第 1 章 绪论

1.1 研究背景与意义

居民消费和出行是当前经济社会发展的核心议题之一。《扩大内需战略规划纲要（2022 － 2035 年）》强调，构建完整的内需体系是推动经济高质量发展的必然选择。在此背景下，人工智能技术，作为新一代信息技术的代表，将在促进消费和出行领域的智能化转型中发挥关键作用。

近年来，人工智能在经济发展和社会治理中展现了广泛的应用潜力。《中华人民共和国国民经济和社会发展第十四个五年规划和 2035 年远景目标纲要》明确提出，未来十年内，人工智能将在关键核心技术方面取得重大突破，推动智能化在各个领域的应用。” 十四五” 期间提出的三大人工智能发展布局——突破核心技术、打造数字经济新优势、营造良好数字生态——为消费和出行领域的智能化升级奠定了坚实的基础。

随着大语言模型（Large Language Model, LLM）的快速发展，其在消费和出行领域的应用逐渐凸显。大语言模型能够处理海量数据，具备强大的自然语言理解与生成能力，为智能化工具的集成提供了技术支持。通过将 LLM 与消费、出行等领域的工具集成，能够有效提升用户体验和服务效率。例如，基于大语言模型的智能体可以调用多种消费和出行相关的 API，自动识别用户需求，为用户提供个性化的决策支持和出行规划。

在此过程中，知识图谱的角色也发生了显著转变。过去，知识图谱主要用于构建领域内的结构化知识，建立实体及其关系网络，帮助系统实现更加精准的信息检索和问答。它依赖专家手工构建或自动化系统提取的知识，能够为特定领域中的推理提供逻辑支持。知识图谱的强项在于关系推理、知识验证以及为复杂任务提供可解释性。

如今，在大语言模型的语境中，知识图谱不再仅仅是孤立的信息源，而成为大语言模型的重要补充和增强工具。大语言模型具备广泛的语义理解和生成能力，但由于缺乏严格的逻辑结构和推理框架，有时会出现逻辑不连贯或生成内容不够精准的问题。而知识图谱可以弥补这一不足：通过将结构化的知识与大语言模型结合，知识图谱能够为大语言模型提供精准的背景信息与逻辑推理能力。具体而言，在智能体工具利用领域，知识图谱可以帮助大语言模型在复杂任务中识别工具之间的关系，辅助工具调用路径的优化，并增强模型的可解释性和推理能力。

尽管大语言模型在工具调用和集成方面展现了显著的潜力，仍然存在以下关键问题：

- 无法集成大规模工具：当前的系统虽然可以调用多种 API，但面对大量的工具集成时，表现出局限性。大语言模型需要处理不同工具的数据格式、调用方式以及响应速度，当工具数量较大时，模型的调用能力和协调效率会明显下降，难以实现对大规模工具的无缝集成。这种限制影响了系统在复杂场景中的适应性与灵活性。
- 难以处理工具之间的依赖关系：对于涉及多个工具协同工作的任务，模型在处理工具间的依赖关系时往往力不从心。例如，在消费推荐和出行规划的场景中，某些工具的输出需要成为下一个工具的输入，且各工具的调用顺序和逻辑关系十分重要。目前，大语言模型难以有效管理这些复杂的依赖关系，容易导致任务流程的中断或结果的误差。

1.2　国内外研究现状

本节将会概述针对大语言模型智能体和知识图谱的研究进展。首先，本节会介绍大语言模型智能体工具调用方面的研究，分为基于提示词工程的方法和基于模型微调的方法。其次，本节将聚焦于大语言模型规划、推理提升的方面，包括提示词工程和模型微调的方法。最后，本节还会介绍知识图谱和大语言模型结合的应用，比如如何利用知识图谱中的外部知识增强大语言模型，以及如何在图谱上进行推理。

1.2.1　大语言模型智能体工具调用

外部工具的引入不仅能够增强大语言模型的能力，比如获取最新知识、提升专业技能，流程自动化、交互增强等。同时，外部工具的采用也能够提高生成过程中的透明性和鲁棒性，让回答更加可靠和可解释。在大语言模型智能体工具调用领域，研究人员已经提出了许多方法来实现大语言模型工具调用。总体来讲，许多工作中的大语言模型工具调用流程都包含以下四个阶段：任务规划、工具选择、工具调用和响应生成[1-3]。

本文主要聚焦于任务规划和工具选择的部分，即如何根据用户需求从众多工具中选择合适的一个或一组工具来形成工具调用链来完成用户的需求。

1.2.1.1 工具任务规划

在现实的信息查询场景中，用户的查询需求往往包含复杂的意图，如何识别用户意图是工具调用的首要问题。因此在工具任务规划阶段，我们首先需要将用户的需求语句转化为更加明确的任务，进行子任务的拆解和任务之间的关联分析。任务规划方式一般分为基于提示词工程的方式和基于微调的方式。

基于提示词工程的工具规划 现有研究[4]表示，大语言模型能够通过少样本甚至零样本实现有效的任务规划。HuggingGPT[2]首先把任务分解为各种子任务，然后选择合适的模型来解决这些子任务。RestGPT[3]引入了一种从粗粒度到细粒度的规划方法，能够指导大语言模型逐步对任务进行分解。ControlLLM[5]引入了一种叫做“在图上思考”（Think-on-graph, ToG）的范式，通过深度优先搜索算法（DFS）在构建好的工具图上进行搜索，得到解决方案。

基于微调的工具规划 Toolformer 通过工具调用来辅助模型预测后续词元，基于此原理对模型进行了微调，从而提升大语言模型的工具认知和工具调用效率。TookenGPT 中将工具作为了特殊词符，以生成普通文字输出的方式对外部工具进行调用。α-Umi 提出了一种新的两阶段的训练模式，首先对基础大语言模型进行较为通用的微调，随后细分为规划器、调用器等模块，分别进行更加针对性的微调。

1.2.1.2 工具选择

在任务规划阶段完成后，需要根据每个子任务进行工具选择。工具选择过程一般有两种途径：一种是通过训练得到的检索器来选择工具，另一种是直接让大语言模型从工具列表中选择合适的工具。

基于检索器的工具选择 当工具数量过多时，通常会使用检索器先搜索得到与任务相关的工具。检索方式包括基于关键词的检索和基于语义的检索两种。

1. **基于关键词的检索**：如 TF-IDF[6]和 BM25[7]。这些方法通过精确匹配实现用户需求和文档之间的对齐和查询。
2. **基于语义的检索**：利用神经网络来学习文本之间的语义关系，然后使用余弦相似度等算法计算语义相似度，如 ToolLLM[8]。

基于大语言模型的工具选择　在工具数量有限，或者是已经检索得到少量有关工具时，可以让大语言模型利用自身的推理和分析能力选择最合适的工具。具体来说，我们可以将备选工具的工具名称、工具描述信息和参数列表与用户需求一起放入大语言模型的输入上下文，提供给模型。随后，模型根据用户需求选择合适的工具。现有的基于大语言模型的工具调用方法分为两类：基于提示词工程的方法和基于模型微调的方法。

基于提示词工程的方法：该方法利用大语言模型的上下文学习能力，通过编写提示词来进行工作。有一些通用的提示词技巧可以帮助在多个工具中选择正确的工具。Chain of Thought（CoT）[9]在提示词中加入了例子，让大模型在解决复杂问题时采取相应的推理步骤，让大模型以分步的方式来规划和行动。Re-Prompting[10]在生成计划之前会检查每个步骤是否能够执行。如果不能够执行，则让大模型重新生成计划。Self-consistent CoT（CoT-SC）<empty citation>因此让大模型执行多条推理路径，选择出现频率最高的答案输出。Tree of Thoughts（ToT）[11]用树状的形式组织推理过程，树上的每个节点表示一个“想法”即推理中间步骤。中间步骤的选择基于大模型的评估，最终计划用深度优先遍历（DFS）或者广度优先遍历（BFS）得出。在 GoT[12]中，作者把用树状结构组织推理扩展为了用图结构组织。

引入环境反馈同样可以提升能力，ReAct[13]中指导大模型按照 thought-action-observation 的方式来解决问题。生成的想法来帮助大模型进行推理和规划，基于这个想法大模型会采取不同的行动，最后观察该行为的结果并作为反馈提供给大模型。Voyager[14]里智能体接收的反馈包括三种：程序执行的中间结果、执行错误描述和自我验证结果。Inner Monologue[15]主动获取人类的反馈，将其与环境反馈进行结合，用于增强大模型的规划和推理能力。SelfCheck[4]则让智能体对自己的推理步骤进行检查和评估，根据结果来修改计划以提升性能。

关于专门针对工具选择场景的提示词工程和流程搭建工作，ToolNet[16]将大量工具组织成为有向图的形式，允许大语言模型从初始节点出发，迭代地在图上选择下一个工具，直到完成任务。ToolLLM[8]中提出了基于深度优先遍历算法的决策树算法，通过支持回溯操作解决了在工具选择上的错误传播问题，有效提高了整体的准确性和通过率。AnyTool[17]提出了一种自我反思的层次化选择的方法，通过在结构化的工具调用树上迭代选择合适的工具。

基于模型微调的方法：该方法通过对大语言模型进行参数微调，以提高其在工具选择中的表现。此类方法通常涉及到利用额外的训练参数或者针对性训练，从而增

强模型的工具选择能力。ToolLLaMA[8]利用在 ToolBench 数据集中 DFSDT 算法所得到的的指令-推理轨迹对微调了 LLaMA-7B 的模型，有效增强了开源大模型的工具能力。ToolAlpaca[18]提出了一种自动化构建工具调用数据的框架，构建了 3.9K 条工具调用数据集微调得到了 ToolAlpaca-7B 和 ToolAlpaca-13B 的模型。ToolVerifier[19]提出了一种“自我验证”的思想，通过在工具选择过程中自问自答一组问题来区分相似的候选工具。不管是基于提示词工程还是基于模型微调的方法，都有各自的优缺点和特点，但是都能够有效提升模型的工具选择能力。基于提示词工程的方法不需要对模型参数进行修改，通过精心构建的提示词构建和流程搭建来提升大语言模型在工具选择中的能力，并适用于所有的大语言模型。而基于模型微调的方法相对复杂，并且仅能适用于开源大语言模型。在微调中需要消耗计算资源，通过调整参数的方式将工具有关的知识注入到模型中。

1.2.2 大语言模型与知识图谱结合的应用

尽管大语言模型主要用于纯文本的场景，但是在许多现实场景中，文本数据与丰富的结构信息以图谱的方式存储。此外，大语言模型的基于文本的推理能力已经得到较多的展现，但是大语言模型在图谱上的推理能力仍有很大的探索空间。

通过将现实世界的知识表示为结构化的知识图谱，并在图谱上进行推理和演算，能够解决许多重要问题。

关于如何将图谱上的知识提供给大语言模型的问题，有三种常见的方法：

1. 自然语言描述。用自然语言描述图结构是最简单的方式，可以直接描述图上的边和邻接列表。
2. 对图进行文本上的改写。由于自然语言描述图通常会较为复杂，而且不具备结构化的特点，因此对图的描述进行了改写，得到了更加高效的图的描述，有利于模型对图谱信息的利用。
3. 对图进行编码。最后一种方法是通过训练图编码器，将图结构编码成为特征序列并作为特征的一部分输入到大语言模型中。这种方法涉及到对大语言模型的微调，以让其适应新的输入格式。

通过将图谱的信息通过自然语言文本或者嵌入向量的格式输入到大语言模型上，可以在图谱上进行推理和搜索。常见的做法是利用深度优先搜索算法（DFS）或者广度优先搜索算法（BFS）来实现在图上的推理和搜索。许多研究探索了基于搜索的推理，特别是在知识图谱领域。Reasoning on Graph[20]的方法将知识图谱作为可靠的知识来源，通过提示大语言模型生成多个关系路径作为计划，随后根据这些路径不

断在知识图谱上搜索，有效地提高了回答的可信度和效率。另一种方法是在图谱上动态地进行迭代检索和推理子图来模拟动态搜索过程[16,21-22]。在每个步骤中，大语言模型都检索当前节点的邻居节点，然后决定下一步操作是继续搜索还是结束搜索并给出答案。在 ToolNet[16]中，作者根据工具调用的数据集建立了图谱，并根据图谱上的边进行搜索，迭代式选择所需的工具进行调用，有效提升了工具搜索的准确性。Think-on-Graph 系列[21-22]在知识图谱上通过大语言模型 Agent 进行迭代式的束搜索，探索发现最好的推理路径，并返回最有可能的推理结果。

这些方法的优点在于，通过图谱的辅助，系统不仅能够提供对应的答案，还可以提供图谱片段作为可以解释的证据。同时，由于知识图谱存储的数据量可以扩展，这些方法通过在图上搜索也增加了系统的可扩展性。

1.3　研究问题

基于以上的研究背景，本文的核心研究问题归纳如下：

- **如何通过工具调用轨迹数据构建一个高效且结构化的工具知识图谱？**本问题旨在研究如何筛选、清洗现有的工具调用轨迹数据，并构建一个大型工具知识图谱，从而将轨迹中的工具知识有效地嵌入到图谱中。重点探讨数据清洗的标准和方法、知识图谱构建的技术路径，以及如何使该图谱在后续的工具搜索和优化流程中发挥有效作用。
- **如何结合知识图谱与智能体架构实现自动化工具编排和调用？**本问题关注如何基于知识图谱和智能体架构，设计一个完整的任务分解、工具选择、调用以及结果解析的流程框架。研究重点包括：智能体如何高效利用知识图谱进行工具调用路径的优化，如何在真实 API 环境下验证智能体编排的有效性，以及如何提升工具调用的精确性和任务完成效率。

1.4　研究内容

本文主要研究基于 Agent 与图谱的任务编排工具的设计与实现流程，主要针对用户进行信息查询时的便利。本文研究与实现的内容主要包括以下几点：

1. 本文提出了一种基于工具调用轨迹数据构建大型工具知识图谱的方法，旨在辅助智能工具搜索与调用流程。首先，通过两种数据筛选策略，筛选出高质量的 API 工具轨迹数据。接着，设计了工具点权和边权值的计算方法和动态更新方法，以表示工具之间的转换关系和工具可用性。最后，通过对比包含图谱知识

与传统检索增强方式，验证了工具知识图谱在实际应用中的有效性。

2. 为了有效利用构建的工具知识图谱的知识，本文提出了基于 DFS 的工具动态搜索算法。该方法通过在图谱上进行深度优先搜索和动态回溯，能够充分利用图谱上的知识选择工具路径。同时，我们提出了长短期记忆框架，通过维护长短期记忆知识，进一步辅助大语言模型的规划和推理。通过在真实世界的 API 测试集上进行实验，验证了该方法的有效性，并且证明该方法优于基线方法。
3. 本文设计并实现了一个基于知识图谱和大语言模型的智能 API 编排与调用系统，旨在提供用户友好的交互体验。本工作允许用户通过自然语言提问，系统能够解析需求并自动调用相关 API，生成答案。其主要功能包括用户登录、API 调用流程可视化、问答服务以及自定义工具添加，而管理员可以管理模型超参数配置和数据库。系统的整体架构分为存储层、访问层、功能层、接口层和展示层五层，各层负责不同的功能，以确保系统的高效性和易用性。

1.5 论文组织结构

本论文的内容组织结构分为以下几章：

第一章为绪论，本章节从研究背景出发，依次介绍了本文研究的问题以及难点，国内外的研究现状，以及本文的主要工作内容以及本文组织结构。

第二章为相关理论与技术，本章主要介绍了有关的一些概念以及现有的技术方案等。首先介绍了知识图谱的概念以及分类，基于 API 知识图谱的研究。其次介绍了大语言模型的定义、发展历史以及大语言模型有关的技术：大模型智能体、提示词工程、检索增强技术等。最后，本章介绍了有关大语言模型和知识图谱结合的应用方案。

第三章为基于工具调用轨迹的知识图谱构建方法及实现。本章主要介绍了如何针对现有的工具调用轨迹数据进行筛选和清洗，并根据其构建一个大型的工具知识图谱。通过边权和点权的更新来表示工具知识，以辅助后续的工具搜索流程。

第四章为基于知识图谱和智能体的工具编排和调用方法的设计与实现。本章聚焦于大模型工具任务的各阶段，比如如何完成任务分解、工具选择、工具调用、结果解析等流程。并在真实世界的 API 工具上设计测试集验证了该方法的有效性。

第五章为系统设计和实现。对系统的架构以及各功能模块进行了设计与实现，在实现前几章各模块的基础上实现了可视化界面和流程搭建。具体内容包括有系统框架设计，系统功能模块介绍以及系统展示等。

第六章为总结与展望，对全文的研究工作进行了回顾，总结了成果，并且对本研究的局限性和未来的研究方向进行了展望。

第 2 章 相关技术分析

2.1 知识图谱

2.1.1 知识图谱的概念

知识图谱是一种对于现实世界中的知识和概念的建模。知识图谱的概念最早由谷歌于 2012 年提出，最初旨在用于构建更智能的搜索引擎。然而，如今知识图谱已经成为人工智能领域中常用的知识表达和存储技术，能够将大量的非结构化知识组织成结构化形式，并被广泛应用于搜索引擎、问答系统和推荐系统等人工智能应用中。

在知识图谱中，知识以图形、结构等可视化方式存储，其中图中的节点代表实体，用于描述真实世界中的物体或抽象概念，而节点之间的边则表示实体之间的语义或逻辑关系。知识图谱的基本组成单位是三元组，包括头实体、关系和尾实体。这种基本结构为知识图谱的构建和应用提供了坚实的基础。

2.1.2 知识图谱的种类

知识图谱的发展大致经历了四个阶段[23]：

1. **静态知识图谱**: 早期的知识图谱大多都用于存储静态知识，其中的三元组不被更新或不常被更新。
2. **动态知识图谱**: 为了保证知识的实时性，知识图谱需要定期被修改或更新。
3. **时序知识图谱**: 时序知识图谱中加入了时序信息的表示，能够提供一种更全面的在时序上了解知识的方式。
4. **事件知识图谱**: 事件知识图谱的重点在于如何表示和理解事件。事件会涉及不同的实体和关系，而且在特定的时间段发生，这让事件表达变得格外困难。事件知识图谱对图谱的结构进行了修改，加入了表示事件的节点和两种不同的关系形式（实体-事件关系和事件-事件关系）。

2.1.3 API 知识图谱

本研究主要针对的场景是通过将不同工具进行组合来完成特定任务。API 有关的知识通常存在于不同信息源中，比如官方参考文档、问答网站等非结构化的文本中[24]。同时，API 之间的依赖关系属于过程性知识。过程性知识是一种关于“怎么做”

的知识，是一种没有明确提取线索，只能借助某种活动形式间接推出来的知识。在本研究的场景中，我们需要把 API 有关知识提供给大语言模型，让模型针对任务描述选择合适的 API 和 API 调用顺序。

由于大语言模型上下文长度有限，难以把所有的 API 相关信息都以提示词的方式提供给大语言模型。而且大语言模型本质上还是一个黑盒，在输入了 API 有关信息之后，无法预测它的输出是否涵盖了我们需要的 API 知识，以及这些知识是否准确、全面。而且，API 之间的依赖关系在任务编排时十分重要，如何正确表达并查询 API 的依赖关系是本研究的关键问题之一。

知识图谱能够结构化地表示信息之间的语义关联，能够用于 API 的知识表示。**Liu2019**中，作者们从 API 参考文档和维基百科中提取相关知识，构建 API 知识图谱。[25]中从 API 参考文档和 API 教程中提取警告语句，构建 API 警告知识图谱。[26]以开源项目中的 API 为实体，以 API 之间的调用、返回、实现等为关系，构建基于开源项目的 API 知识图谱。[27]中，作者设计了一个仅包含三种实体的知识图谱，在图上使用随机游走等算法来实现 API 的推荐。

2.2　大语言模型

2.2.1　大语言模型的定义

大语言模型（LLMs）主要指基于 Transformer 架构的语言模型，通常具有数十亿到百亿个参数。LLM 通过在海量文本数据上进行预训练，学习词汇、句法、语义和语境之间的关系，能够生成和理解复杂的自然语言。形式化定义如下：

设有一个语言模型 $\mathcal{M}$，它通过给定一个输入序列 $x = (x_1, x_2, \ldots, x_n)$ 来预测下一个词或生成输出序列 $y = (y_1, y_2, \ldots, y_m)$。模型的目标是最大化给定上下文时生成词的联合概率分布 $P(y_1, y_2, \ldots, y_m | x_1, x_2, \ldots, x_n)$。这个联合概率可以通过链式法则表示为：

$$P(y_1, y_2, \ldots, y_m | x_1, x_2, \ldots, x_n) = \prod_{i=1}^{m} P(y_i | y_1, \ldots, y_{i-1}, x_1, x_2, \ldots, x_n)$$

其中，$P(y_i | y_1, \ldots, y_{i-1}, x_1, x_2, \ldots, x_n)$ 是模型在给定先前上下文和输入序列时对 y_i 的条件概率预测。

随着大语言模型参数量和规模的提升、以及训练文本量的增大，大语言模型展现出了小模型不具有的“涌现能力”。比如：（1）上下文学习能力，大语言模型能够从提

示词中提供的小规模样本中学习如何完成新任务。(2) 指令遵循能力，在经过指令微调后，大语言模型能够遵循新任务的指令。(3) 复杂推理能力，大语言模型能够通过将复杂任务分解为中间推理步骤来解决复杂任务。大语言模型还可以通过外部知识和工具调用来增强，以便获得更强大的能力和提升任务的准确性和可靠性。

现有的大模型根据是否开源可以分为两种，第一种是闭源的商业大模型，比如：ChatGPT，GPT-4[28]，Claude-3.5，Gemini-2 等等，用户和开发者可以通过官方提供的平台或者 API 接口来使用这些模型；另一种是开源的大模型，比如：Baichuan[29]，ChatGLM[30]，Qwen，LLaMA[31]等。这类模型的效果一般比商业大模型的效果弱一些，但由于是开源的，开发者可以根据具体需求构建数据集对模型有监督的微调等进一步参数调整。

2.2.2 大语言模型提示词工程

提示词工程是通过创建自然语言指令（即提示词）来从大型语言模型中提取知识的过程，已成为提升预训练大型语言模型能力的重要技术之一。通过精心设计提示词，可以在不更改模型参数的情况下提高模型输出的表现。与传统方法相比，提示词工程无需对模型进行训练或微调即可实现特定任务上的性能提升，从而有效增强大型语言模型在不同领域的适应性和可用性。

目前提示词工程的技术体系十分多样化，涵盖了从最基本的零样本提示（Zero-Shot Prompting）和少样本提示（Few-Shot Prompting）到更复杂的“思维链提示”（Chain of Thought Prompting）等多种方法。

- **零样本提示（Zero-Shot Prompting）**：在零样本提示设置（Zero-Shot，Radford 等，2019）中，大型语言模型完全依赖于在预训练过程中学到的知识，通过提示词中的指令直接执行任务，而无需任何额外的示例数据。该方法的优点在于操作简单，但在任务理解和推理复杂度上往往会受到限制。
- **少样本提示（Few-Shot Prompting）**：在少样本提示设置中（Few-Shot，Brown 等，2020），为了更好地理解任务，除了提供任务指令，还会加入少量的示例数据点来帮助模型掌握上下文语境和任务要求。研究表明，精心设计的少样本提示能够显著提升模型的性能，但如果选取的示例不当，模型可能会对这些样本产生难以预料的偏差。少样本提示策略在提示词工程中被视为一种有效的方式，用于提升模型的规划和推理能力。

以下是几种针对提升大型语言模型在复杂推理任务中的能力而提出的提示策略：

- **基本提示（Basic/Vanilla Prompting）**：基本提示是最基础和最简单的提示词策

略，指的是直接向语言模型提供任务，而不进行任何提示词策略的优化。该方法的目标是评估模型在没有提供外部信息时的性能表现。在不同的研究中，基本提示也被称为“标准提示”或“原始提示”，常作为各类提示策略的基础对比。

- **思维链提示（Chain-of-Thought，CoT）**：[9]提出了一种思维链提示策略，该策略通过将复杂问题分解为更容易理解的子问题，然后逐步推理并整合各个子问题的解答来得出最终答案。这种方法模拟了人类在解决问题时的逐步推理过程，展示了显式推理链条对复杂任务的性能提升效果。
- **思维树提示（Tree-of-Thought，ToT）**：该提示词框架在思维链的基础上进行扩展，以树状结构管理中间推理步骤，进一步增强了大语言模型在面对复杂任务时的推理能力。思维树结合了模型生成和评估“思维”的能力，并使用了一些常见的搜索算法，如深度优先算法和广度优先算法来在树上进行搜索。在思维树框架下，模型可以系统化地对不同推理路径进行探索，并且在出现错误时能够及时回溯。
- **自我一致性提示（Self-Consistency）**：自我一致性提示通过生成多个回答并选择出现频率最高的答案来提高思维链的表现，有利于提升推理的准确性和一致性。
- **ReAct 提示词**：该策略通过将复杂问题分解为更容易理解的子问题，然后逐步推理并整合各个子问题的解答来得出最终答案。在每一步，系统都会生成思维（Thought）、行为（Action）和观察（Observation）三部分的内容，并加入大语言模型的上下文来提示模型进行推理。

2.2.3　大语言模型智能体

大语言模型也可以作为“智能体”来为用户提供服务。智能体的概念的最早可以追溯到古希腊时期的哲学家亚里士多德和休谟等人[32]。从大体上来讲，智能体可以被定义为“含有欲望、信念、动机和行为能力的实体”[33]。后来，计算机科学中也用到了智能体这个概念，在人工智能领域中的 AI 智能体就用来描述具有自主性、主动性、反应性和智能行为能力的实体[34]。早期的 AI 智能体研究主要集中在增强智能体的特定能力上[35]，并通过改进相应算法和训练策略来提升它们的表现，而忽略了模型本身如记忆、推理能力的综合能力的提升。然而，模型本身的能力很大程度上决定了智能体在任务上的表现。

近年来，随着大语言模型的出现，人们发现它们在许多任务上都取得了出色的成绩。大模型具有庞大的模型参数，并且经过在大量数据上的训练，这使得它们具有强

大的知识获取能力、规划和推理能力以及泛化性，能够很流畅地与用户进行交互[36]。这些能力在 AI 智能体中非常有用，因此衍生出了许多基于大模型的智能体研究，使用大语言模型作为智能体的中央控制器，通过感知环境的变化和不断做出决策，能够很好地解决多种复杂任务。为了使得大语言模型能够不断做出决策和感知环境的变化，搭建大语言模型智能体通常需要使用外部知识获取、工具调用等增强技术。为了让大语言模型的能力在智能体中得到充分发挥，研究者们设计了不同的模块和架构。OpenAI 科学家 Lilian Weng 在博客[37]中提出了一个统一的大语言智能体的架构，包含记忆、任务编排和工具使用三个关键模块。

- **记忆**：记忆模块是大模型的整体架构中非常重要的一部分。记忆包括从外界环境感知到的信息和记录在知识库中的记忆，能够指导大模型作出更准确、更快速和更具有一致性的行为。在进行模块设计时，研究者们参考了人类的记忆方式，因此大模型智能体的记忆方式类似人类的短期记忆和长期记忆：大模型智能体的“短期记忆”常常指的是 Transformer[38]架构的上下文窗口输入的内容；而“长期记忆”则用来表示大模型可以随时查询和获取的外界知识库。**Wang2023**中将大模型常用的记忆方式分为两种：仅使用短期记忆，和长短期记忆混合方式。文中还提到了，由于大部分智能体都需要动态地感知环境的变化，并对环境做出连贯的反应，大部分智能体的实现都要使用短期记忆，因此本文不讨论仅使用长期记忆的模式。
- **任务编排与规划**：在处理复杂任务时，将其分解为更简单的子任务是一种有效的方式。大语言模型的规划模块就希望能够让大模型也具有这样的分解子任务的能力。本文将规划模块的实现方式根据是否有反馈分为两种。无反馈的规划模块一般通过不同的提示词工程技巧、或者不同的路径搜索算法来提升整体的任务规划能力。在复杂且多变的现实场景里，在没有反馈时很难直接生成正确可执行的计划，而在有反馈的规划方式中，智能体在行动后可以得到环境、人类及模型的反馈，并据此修改现有计划，以得到更好的执行结果。
- **工具使用**：大模型本身具备丰富的内部知识，因此在行为部分，大语言模型既可以使用自身能力的理解任务、做出规划、完成任务，也可以通过外部加入的工具来进一步扩大模型的行为空间。大语言模型可以很好地理解外部工具的作用，并在合适的时候调用工具并对工具返回的结果进行处理，得到最终结果进行输出。

2.2.4　大语言模型检索增强生成

检索增强生成（RAG）是一种通过结合外部知识库而提升大语言模型能力的一种技术。检索增强技术一般用于知识密集型技术，能够通过相似度检索的方式从外部领域知识库进行检索，从而提升特定任务的准确性和可信度。通过这种方式，可以将模型内部的知识与庞大的、动态的外部知识库进行有机结合，扩大大语言模型的能力范围。

检索增强生成的流程一般有以下三个阶段：索引、检索和生成。索引阶段从对多种格式的原始数据进行清理和提取开始，随后将这些数据转换为统一的纯文本格式。为了适应大语言模型的上下文限制，文本会被划分成较小、易于处理的块。接下来，这些块通过嵌入模型被编码为向量表示，并存储在向量数据库中。这一步骤对于后续检索阶段的高效相似性搜索至关重要。

整个检索增强生成系统由两个核心模块组成：检索器和生成器。检索器从数据存储中搜索相关信息，生成器则生成所需内容。检索增强的过程如下所示：（1）检索器最初接收到输入查询，并搜索相关信息；（2）然后，原始查询和检索结果通过特定的增强方法输入到生成器中；（3）最后，由生成器生成所需的输出内容。

在不同的应用中，我们可以使用不同的生成器模块。在本文我们讨论的是基于大模型的任务，因此生成器为大语言模型。而检索器模块的作用是在给定需求信息的情况下识别并获取相关信息。主流的检索方法可以分为稀疏检索、密集检索两种。不管是稀疏检索还是密集检索，检索的过程可以分为两个阶段：（1）将每个对象编码为特定的表现形式（2）构建索引来对这些搜索对象进行高效检索。

检索的目的是在给定信息需求下识别并获取相关信息，可视为从键值存储中找到最相似的键并获取对应的值。检索方法主要分为稀疏检索和密集检索：

- **稀疏检索**：常用于文档检索，利用词匹配度量（如 TF-IDF、BM25）分析文本词频，构建倒排索引进行高效搜索。它也应用于知识图谱，通过关系连接实体，支持 k 跳邻居搜索或命名实体识别（NER）。
- **密集检索**：通过密集嵌入向量表示查询和键，使用近似最近邻（ANN）索引加速搜索，适用于文本、代码、音频、图像等多种数据模态。模型使用对比学习优化检索效果，并利用树结构、局部敏感哈希等索引技术提升搜索效率。
- **其他方法**：一些方法使用自然语言文本的编辑距离直接进行检索，而不计算嵌入表示。在知识图谱中，实体通过关系相连构成图，这些实体之间的关系也相当于预先构建的检索索引。因此，基于知识图谱的检索增强生成方法可以采用

k 跳邻居[39-40]。

通过检索器的检索，系统得到了与输入信息最为相似的前 K 个块，这些块将作为扩展上下文用在大语言模型的提示词中。所提出的查询与检索得到的会被整合成一个连贯的提示，以便请求大型语言模型生成响应。模型的回答方式可能根据特定任务的标准而有所不同，它可以依赖于自身的参数知识或限制其响应内容仅来自所提供文档。

知识图谱也逐渐被集成到检索增强系统中。知识图谱以三元组的形式存储实体及其关系，能够以更紧凑的方式表达世界知识。相比于文档，知识图谱提供的知识更加结构化、明确，可以帮助减少冗余信息。在问答任务中，使用 KG 的检索增强生成实现可以显著提高最终结果的正确率，大语言模型可以很好地利用知识图谱中搜索得到的结构化准确知识，生成准确性、鲁棒性更高的回答。

2.2.5 知识图谱与大语言模型相结合

大语言模型是在大规模语料库上预训练得到的，它们在许多自然语言处理任务上都展示出不错的效果。随着模型的训练数据规模和参数规模的增大，大语言模型能够完成更多复杂的任务。然而，大语言模型也有许多的局限性。它们的知识范围仅限于训练时用到的语料库[41]，无法对知识进行及时的更新。并且大语言模型在很多时候会生成一些与事实不符的回答[42]，即幻觉现象。在许多专业的领域，幻觉现象极大地限制了大模型的应用。除此之外，由于计算资源和成本的考虑，大语言模型的上下文长度受限，对长输入的处理仍然是一个问题。

将知识图谱引入大模型能够帮助解决这些问题，通过引入外部的知识图谱知识，可以通过动态更新图谱的方式来引入最新知识。此外，将庞大的知识库转化为结构化的知识图谱后，每次可以根据不同需求进行搜索，得到，来保证准确可靠的领域知识。如何有效地对现实世界的事实进行建模是一个关键问题。现有的图谱构建方法通常由人工构建，缺乏足够的泛化能力，因此有必要利用大语言模型来辅助图谱的构建过程。

将大语言模型和知识图谱联合起来的方式能够同时增强它们两者的能力。在知识图谱增强的大语言模型中，知识图谱既可以在预训练和推理的时候提供知识[43]，又可以增强大语言模型的可解释性[44]。在大语言模型增强的知识图谱中，大语言模型可以用在知识图谱的不同任务中来辅助知识图谱的应用。而在大语言模型和知识图谱的融合系统中，研究者们将大语言模型和知识图谱的优点相结合，用于增强知识表达[45]和推理[46]。

2.3　向量嵌入模型

2.3.1　向量嵌入模型介绍

2.3.2　微调方式

2.4　工程技术

2.4.1　Neo4j 图数据库

Neo4j 是一种高性能的 NoSQL 图数据库，最早于 2003 年开发，并于 2007 年发布。作为当前领先的图数据库之一，Neo4j 基于属性图模型，能够以键值对的形式存储节点和节点之间的关系，极大地增强了图数据模型的表现能力。其专属查询语言 Cypher 具备直观、高效的特点，方便用户对图数据进行快速查询和操作。

Neo4j 采用原生图形处理引擎（GPE），具备完整的 ACID 事务支持，确保了数据操作的原子性、一致性、隔离性和持久性。此外，它提供了 REST API，使得用户能够通过多种编程语言方便地访问数据库，支持两种 Java API：Cypher API 和原生 Java API。这使得 Neo4j 在处理连接密集型数据时具有显著优势，尤其适用于表示半结构化数据、快速检索和导航复杂关系网络。Neo4j 的数据浏览器还支持将查询结果导出为 JSON 或 XLS 格式，为用户提供了灵活的操作和集成能力。

2.4.2　Qdrant 向量数据库

Qdrant 是一款开源的高性能向量数据库，专为下一代 AI 应用而设计。它采用云原生架构，并通过 RESTful 和 gRPC API 支持向量的管理和检索。Qdrant 的核心特点在于其高效的高维向量存储和查询能力，特别适用于语义搜索和推荐系统等场景。通过将向量嵌入与附加的元数据结合，Qdrant 提供了更灵活的过滤和搜索选项。

该数据库能够支持数十亿个数据点的存储与查询，同时具备实时分析的能力。在性能方面，Qdrant 采用先进的索引技术，例如层次式可导航小世界（HNSW），实现高效的近似最近邻搜索。用户可以根据具体需求选择多种相似度度量标准，包括欧式距离、余弦相似度和点积。这些度量标准确保了 Qdrant 在相似性搜索中既快速又准确，有效满足 AI 和机器学习应用的需求。

2.4.3　LangChain

LangChain 是一个专为开发大型语言模型（LLMs）驱动的应用程序而设计的框架，旨在简化应用程序的整个生命周期，从开发到生产化的各个环节。LangChain 提供了一系列开源构建模块、组件及第三方集成，方便开发者构建应用。这些核心库包

括基本抽象和 LangChain 表达语言，以及与第三方服务的集成，使开发者能够高效构建应用的认知架构。

LangChain 的组件具有模块化和易用性，开发者可以选择是否使用整个框架。内置的现成链简化了入门过程，帮助开发者快速上手，同时也允许灵活自定义现有链或构建新的链，以满足特定的应用需求。

2.4.4 本章小结

本章介绍了大语言模型的基本概念、相关技术与其在智能体、提示词工程、检索增强生成中的应用，并讨论了如何将知识图谱与大语言模型相结合来提升其在不同任务中的表现。

第 3 章 基于工具调用路径的知识图谱构建

3.1 引言

工具调用路径中蕴涵了工具之间的调用关系信息，通过图的格式对改信息进行建模后，能够通过在图上的搜索来获取工具之间的依赖。本章介绍了一种基于工具调用路径数据进行工具知识图谱构造的策略，并基于此方法实现了一个大规模的工具知识图谱。

本章将会围绕着工具知识图谱构建中的一些挑战出发，具体来说：1. 如何对工具调用路径数据进行数据筛选，得到高质量的工具集以及工具调用路径？2. 如何设计工具知识图谱的结构，以表示工具之间隐含的依赖关系？3. 如何验证工具图谱作为知识库的有效性？4. 如何有效地在图谱上搜索相关节点？

基于上述问题，我们进行了以下研究：首先，我们提出了三种数据筛选策略，从 API 工具和 API 调用路径的角度对数据进行了筛选，保留了一批高质量的工具轨迹数据。其次，我们设计了工具动态转移权值和工具静态转移权值两种方式，通过对工具轨迹数据进行计算，得到了工具之间的转换权值以及每个工具的可用性分数，用于后续辅助大语言模型进行工具选择。然后，关于验证工具图谱作为知识库的有效性，我们对比了直接用普通的检索增强方式和包含图谱知识的方式，验证了构建工具知识图谱的有效性和必要性。最后，我们通过对向量模型进行负样本构造和有监督的微调，获得了一个 API 检索器，并对工具检索的召回率进行了充分实验，证明了该模型在节点搜索上的有效性。

3.2 整体流程

在现实场景的工具调用场景中，其实不同类型的工具之前存在隐含的顺序关系。举一个现实生活中的例子，“城市经纬度查询”和“根据经纬度获取实时天气”的工具总是在一起使用。这种存在于工具调用路径中的“过程性知识”对工具规划非常有启发性，能够表示工具之间的调用关系和跳转关系。基于此，我们希望能够通过对工具之间的转换关系进行建模，并将工具的搜索范围限定在一个更精确的空间，以辅助大语言模型工具调用。

我们分析了目前公开可用的通用工具学习数据集 ToolBench、API-Bank，其中包

含大量的多跳工具调用路径。通过分析其中的数据，我们发现每个工具后都只有一小部分后继工具会被调用，这证实了我们的观点，即工具之间存在相互调用的关系。

因此我们通过对开源的工具调用路径数据进行数据筛选、数据清洗等流程构建了一个大型的工具图谱。数据筛选和清洗流程中，本文制定了详细的规则，筛选得到高质量的工具推理轨迹作为图谱构建的基础数据。同时，我们设计了两种不同的图谱构建策略：静态图谱构建和动态图谱更新，两种方式相辅相成，都对图谱上的转换权值和可用性权值进行修改。在最终构建的图谱中，图上的节点为对应的开源工具，图上的边为有向边，标记了工具之间的跳转和调用关系，边上的权值为转换权值，即多有可能从当前工具跳转到下一工具。点上有有可用性权值，表示该工具的可用性，即该工具有多大的可能性能够被调用成功。

在图谱建立后，我们将大量工具信息与工具之间的依赖关系表示与图谱上，并且通过在图谱上进行深度优先遍历或者广度优先遍历等方式选择合适的工具调用路径。关于图谱上的动态搜索算法部分会在下一章详细介绍。

3.3　数据收集和清洗

3.3.1　数据集介绍

目前较为大型的工具调用数据集有 ToolBench[8],API-Bank[47],AgentBench[48]等。

AgentBench 主要聚焦的是多情景的工具调用，比如。而本文侧重点在于真实世界的 API 工具，因此我们基于 ToolBench 和 API-Bank 构建了工具图谱。

ToolBench 包括来自 49 个类别的 16464 个真实的 RestAPI 工具，并根据这些工具构建了包含 126486 条推理轨迹的数据集，共调用了 469585 次 API。ToolBench 中使用的 API 都来自于一个开放的 API 托管平台“RapidAPI Hub”，在这个 API 托管平台中包含各种不同类别、不同供应商的 API 服务，需要订阅了相应的工具集才能够使用。在 RapidAPI Hub 中，工具以以下的方式进行组织，即网站上有不同的大类，大类下面会有不同的工具集，而工具集下面会有对应的单独的 API 工具。

ToolBench 中包含不同类型的数据，如：1. 每个工具集的详细描述信息，包括工具集的名称、描述、类别等 2. 工具的输入、输出参数列表 3. 工具调用的示例代码 4. 通过 GPT-3.5 生成的真实调用路径和参考 API 工具列表等等。

在本文中，为了构建工具知识图谱，因此我们在图谱构建部分主要利用的是通过 GPT-3.5 生成的工具调用路径数据。由于 RapidAPI Hub 上，同功能的工具可能有很多，即 API 工具之间存在着相似性或者能够从功能上相互替换，因此对于同一个

用户任务，有多条路径都能够解决问题。因此对于每个需求，并不存在一个标准答案路径。但是我们可以通过分析最终的回答结果，判断经过一组 API 顺序调用之后系统输出的结果是否符合预期，来判断路径是否合理。

3.3.2 数据清洗

通过大语言模型构建的工具调用路径数据可能会有重复调用/API 工具编排错误等问题，并且数据集中的工具调用是以决策树的方式组织的，需要从中获得正确的工具路径。基于上述问题，我们对工具调用路径数据进行了一些清洗，主要筛选方式如下：

- **删除失败工具调用路径**: 工具调用路径数据最后会对用户需求进行正面解答或放弃，我们只保留成功调用的 API 工具调用路径，删除无法得到结果的工具调用路径。
- **删除过长工具调用路径**: 通过数据分析，我们发现 ToolBench 中的工具轨迹数据的 API 路径长度不一，平均的调用路径长度为 4，但是极个别的 API 工具调用路径长度达到了 20。过长的工具调用路径通常表示提供给大语言模型智能体的工具无法很好地完成该任务，导致工具调用一直在失败，因此我们将工具调用路径数据长度作为筛选指标，筛选掉长度大于 8 的 API 调用路径。因为这些 API 调用路径的 API 工具数量过多，无法进行有效的图谱构建。
- **删除无效工具节点**: 在筛选优质工具轨迹后，我们还需要对工具轨迹数据进行清洗，得到在决策树上的正确的那一条路径。主要分为两种方式：1. 去重，即对于同一个工具，如果有多个 API 的重复调用，我们只保留最后一次该 API 的成功调用记录。2. 删除失败工具，如果工具调用路径中存在工具调用失败的情况，那么我们从轨迹中删除该工具。最后得到的应该是一个线性的调用路径。

原始数据集中包含了 xx 条工具调用路径数据，其中平均的工具调用次数为 xxx 次，经过上述的筛选逻辑后，共剩下 xxx 条工具调用路径数据。我们基于该高质量的轨迹数据集建立工具图谱。

3.4 图谱构建

工具图谱的构建对于系统整体的性能和效率至关重要。在这里，我们参考了 ToolNet[16]中的设计，将 API 工具作为图谱中的工具节点，工具节点之间的边上带有一个权值，该权值叫做“工具转移权值”，用于代表从当前工具出发，有多大的可能性会跳

转到调用另一个工具的使用。除了边上带有权值，我们给工具节点也赋予了一个权值，该权值叫做“工具可用性权值”，用于表示该工具的可用性。

因此，在该工具图谱中，权值的计算方式非常重要。边和点的权值应该有区分度，不然无法提供有效的信息。

我们首先将会介绍基于历史工具调用数据的静态图谱构建方式，由于 API 工具具有时效性和多变性，我们还会对图上的节点、节点权值还有边上的权值进行动态的更新。

3.4.1　静态构建

静态构建图的过程相对简单，只需要遍历轨迹数据，并计算每两个节点之间的边权和点权值即可。

设定 D 为已完成任务的工具使用轨迹集合，每条轨迹由工具使用的顺序构成，

$$\text{Trajectory} = [D_1, D_2, \ldots, D_n]$$

假设每个工具的调用成功与否是一个二元变量 x_i，其中：

$$x_i = \begin{cases} 1, & \text{工具调用成功} \\ -1, & \text{工具调用失败} \end{cases}$$

为了简化操作，我们将“结束”视为一种普通工具，每个工具都与“结束”工具节点相连，当大语言模型智能体认为要结束调用过程时，就会选择“结束”工具。

节点 v_i 到 v_j 的转移权重 $w_{i,j}$ 计算如下：

$$w_{i,j} = \frac{\mathbb{E}_D[1(\text{tool}_s = v_i \wedge \text{tool}_{s+1} = v_j)]}{\mathbb{E}_D[1(\text{tool}_s = v_i)]}$$

其中，$\mathbb{E}_D[\cdot]$ 表示对轨迹集合 D 上的期望操作，$1(\cdot)$ 是指示函数，当条件满足时取值为 1，否则为 0。该公式计算了工具 v_i 被调用后紧接着调用工具 v_j 的频率。

除了关注工具之间的依赖关系，每个单独的工具的可用性和稳定性也是真实场景下重要考虑因素。在工具调用路径中，有一些工具调用会出现无访问权限或访问超时的错误信息。因此我们在工具图中通过对节点的权值进行计算和定义，通过点权来表示该工具的可用性或稳定性。节点 v_i 的可用性权重 α_i 计算如下：

$$\alpha_i = \frac{\sum_{s=1}^{n} 1(x_s = 1 \wedge \text{tool}_s = v_i)}{\sum_{s=1}^{n} 1(\text{tool}_s = v_i)} \quad (0 \le \alpha_i \le 1)$$

这里，α_i 表示工具 v_i 的调用成功率，是该工具在轨迹中成功调用的次数与其总调用次数的比值。因此该权值的范围是 0-1，越接近 1，表示该工具的可用性越高，越接近 0，表示该工具的可用性越低。

3.4.1.1 分数映射函数

函数 $f(x)$ 用于将累积分数映射到 $(0,+\infty)$ 的范围内，从而确保在权重更新过程中生成正值。归一化的必要性在于，它能够确保所有工具的表现有效比较，避免负值对权重更新造成不良影响。具体而言，映射函数确保所有工具的权重更新结果均为正值，这对于构建有效的工具图谱至关重要。同时，函数通过灵活处理正负分数，使算法在不同情境下保持良好表现，保持各工具之间的相对表现，从而提升模型的准确性和稳定性。函数定义如下：

$$f(x) := \begin{cases} \alpha x + 1, & \text{if } x \geq 0 \\ e^{\alpha x}, & \text{if } x < 0 \end{cases}$$

在这个函数中，α 是控制映射函数行为的超参数。对于正分数，映射为线性函数 $f(x) = \alpha x + 1$，使得得分的正向提升以线性方式反映在权重更新中。对于负分数，映射为指数衰减函数 $f(x) = e^{\alpha x}$，这保证了即使在评分较低的情况下，权重的更新也会保持在一个合理的范围内。

通过对筛选后的工具轨迹数据进行上述计算，我们得到了工具图谱的初始静态图谱。

3.4.2 动态更新

基于上述方法构建的静态工具图谱，本文还提出了对边权和点权的动态更新。动态更新对于工具图谱的构建至关重要，原因主要包括以下几点：1. **新工具的初始化问题**：当一个新的 API 工具被引入时，由于缺乏历史调用数据，无法通过传统的静态方法来初始化它的节点信息。因此，在新添加了工具节点后，我们需要对该节点上的边、点权进行不断的更新，以维持最新的状态。2. **用户偏好的动态变化**：在实际应用中，随着用户行为的变化，新的调用路径会不断生成。这些新轨迹反映了用户的真实偏好，因此需要及时添加到工具图谱中，使其能够反映最新的用户需求。3. **工具的生命周期变化**：工具是动态变化的，随着时间推移，有些工具可能会因为开发者的弃用或其他原因而逐渐失去支持。静态构建无法反映这些变化，而需要通过动态更新来调整图中的边权重，确保图谱能够保持对工具现状的准确表示。

动态更新发生的时间节点为系统产生了新的工具调用路径时。

以下是对于边权和点权更新的具体阐述，以及对保持分数为正的映射函数的介绍。

3.4.2.1　边权的动态更新

对图 G 的更新尤为关键。由于可用工具轨迹的数量有限，我们需要对每一条轨迹进行细致检查。LLM 作为工具的评估者，将整个轨迹作为输入，评估每个工具节点的表现。

我们用 $\Delta_i^{(n)}$ 表示在第 n 次迭代中节点 v_i 的评估得分。节点 v_i 的累积得分 $s_i^{(n)}$ 计算如下：

$$s_i^{(n)} = \sum_{k=1}^{n} \Delta_i^{(k)}$$

该公式对节点 v_i 在每次迭代中的评分进行累积，为后续的转移权重更新提供基础。

节点 v_i 到 v_j 的转移权重 $w_{i,j}^{(n)}$ 在第 n 次迭代中的更新公式如下：

$$w_{i,j}^{(n)} = \beta w_{i,j}^{(0)} + (1-\beta)\Delta w_{i,j}^{(n)}$$

其中，$w_{i,j}^{(0)}$ 是初始的转移权重，β 是控制初始权重与动态更新之间平衡的超参数，$\Delta w_{i,j}^{(n)}$ 是用于更新转移权重的归一化梯度，计算公式如下：

$$\Delta w_{i,j}^{(n)} = \frac{f(s_i^{(n)})}{\sum_{v_j \in \text{oneigh}(v_i)} f(s_j^{(n)})}$$

该公式通过累积分数 $s_i^{(n)}$ 和 $s_j^{(n)}$ 来更新转移权重，确保工具之间的转移反映最新的评估结果。

3.4.2.2　点权的动态更新

在点权的动态更新中，我们通过分析新生成的工具调用路径来调整工具的可用性权重。设调用路径为：

$$\text{Trajectory} = [D_1, D_2, \ldots, D_n]$$

对于每个工具 D_i 的调用结果记作 x_i，其中：

- $x_i = -1$ 表示调用出错，
- $x_i = +1$ 表示调用成功。

在这里，出错的调用情况指的是 API 工具本身存在可用性问题，例如遇到状态码“403 Forbidden”、“408 Request Timeout”等。

根据调用结果，我们可以对点权的分数进行更新。更新公式如下：

$$v_i^{(n)} = \beta v_i^{(0)} + (1-\beta)\Delta v_i^{(n)}$$

其中：- $v_i^{(0)}$ 是工具 D_i 的初始权重，- β 是控制初始权重与动态更新之间平衡的超参数，其值会根据错误信息的严重程度进行调整，- $\Delta v_i^{(n)}$ 是用于更新权重的归一化梯度，计算公式如下：

$$\Delta v_i^{(n)} = \frac{f(s_i^{(n)})}{\sum_{D_j \in \text{oneigh}(D_i)} f(s_j^{(n)})}$$

在此公式中，$f(s_i^{(n)})$ 是累积分数，通过反映工具 D_i 的评估结果，确保工具之间的转移权重反映最新的使用情况和反馈。

3.5 本章小结

第 4 章　基于工具图谱与深度优先遍历的 API 编排与调用方法

4.1　引言

在第三章中，我们介绍了基于工具调用路径构建的 API 知识图谱，包括有对调用路径数据的清洗、图谱的静动态构建算法、API 节点召回模型训练等部分。尽管工具图谱能够表示大量的工具调用路径，但是仅依赖图上搜到的路径不具有灵活性，对于不同的用户需求，图上的路径不一定能够灵活满足。同时，工具也具有生命周期和动态性，会有新建的工具或者废弃的工具，图谱上的节点也会随之变化，因此需要一种动态的 API 选择方法。大语言模型能弥补这一种灵活性、动态性的缺乏，但仅依赖大语言模型的选择而不利用图谱信息，则难以解析工具之间的复杂依赖关系。

因此，我们提出了一个在工具图谱上动态遍历的算法，通过大语言模型智能体在图上动态选择节点。本章集中介绍图谱遍历和 API 路径选择的部分，即如何根据用户需求在大型工具图谱上进行高效的搜索和选择。在本章主要有以下几个重要问题需要研究：1. 如何合理利用工具图谱上的依赖信息进行工具选择？2. 如何对路径选择流程进行优化，以提升整体的准确率和效率？3. 如何处理工具调用中遇到的工具调用异常、工具响应过长等问题?

针对上述问题，我们提出了一种基于工具图谱的动态寻路算法。该算法首先将用户的需求进行分析和拆解，以得到更小的任务编排与执行单位。其次，本章提出了一种基于深度优先搜索的搜索算法，能够实时地在图上进行搜索并选择合适的 API 调用路径。最后，由于 API 工具调用的返回结果内容较多，本着减少大语言模型推理时间与提升系统效率的考虑，我们提出了一种响应内容压缩方法，通过让模型选择重要的字段来生成更短的响应结果，能够有效保留重要信息并提升交互效率。

4.2　整体框架

图 4.1展示了本章提出的基于深度优先遍历算法的动态工具编排算法的整体技术框架。

该方法整体由以下几个部分组成：

1. **任务解耦模块**：任务解耦模块负责将复杂、模糊的用户需求解耦为多个涉及不同大类 API 的子任务，能够充分提高系统处理复杂任务的能力。

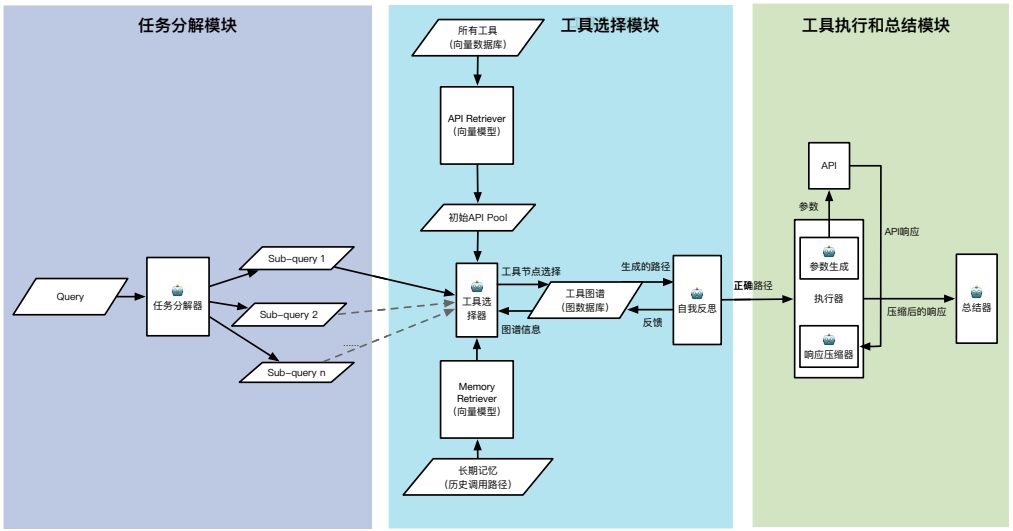

图 4.1　整体框架
Figure 4.1　Overview of the Dynamic Tool Selection Framework

2. **任务编排模块**：该部分包括基于深度优先遍历的动态工具搜索算法、自我反思机制和长短期记忆模块。本质上来说，该算法就是在图上进行深度优先遍历，我们将图上的节点的权值、节点的邻居点的边权等信息，以及节点工具的描述信息等全部提供给大语言模型，让模型在图上动态选择节点。在工具路径选择的过程中，若遇到了路径信息，我们将会进调用“自我反思机制”，即对路径上遇到的错误进行分析并基于此错误分析内容重新进行路径选择。模型可以选择回溯到某个中间过程，或者是从头开始寻路。在大语言模型智能体不断寻路的过程中，我们会维护一个“短期记忆”，即对整体寻路流程的记录。为了利用历史经验知识，我们添加了“长期记忆模块”，即我们会搜索类似任务的历史工具调用路径放在提示词辅助大语言模型的规划。自我反思机制能够通过格式化的反思提升系统的准确性，。
3. **任务执行模块**：在得到了整体的路径后，任务执行模块负责调用工具 API，并将结果进行汇总，最终输出结果。其中涉及到了 API 参数配置、API 响应压缩模块。API 参数配置模块直接将 API 工具的说明信息和所需参数信息提供给模型，让模型提供合适参数并进行校验。API 响应压缩模块则负责对 API 的响应结果进行压缩，以减少响应时间，提升交互效率。
4. **任务总结模块**：任务总结模块即对子任务的工具调用链执行结果进行汇总，输出最终结果。

在流程上，对于每个用户需求，只会执行一次子任务解耦。但是对于每个子任务，都会执行一次动态工具搜索算法，并且会根据搜索到的工具路径执行若干次 API 工具，因此会多次调用 API 执行模块。最后，我们通过任务总结模块对所有子任务的结果进行汇总，得到最终输出。

4.3 具体实现

4.3.1 任务解耦模块

由于用户的需求可能会较为模糊、笼统，或者在同一个需求语句中存在多个潜在的子任务。通过对用户提供的复杂需求进行解耦、改写，并生成适合执行的具体指令，这一过程使得任务变得更加明确和易于解决。

任务解耦模块的输入包括具体用户需求、子任务格式的指令、输出格式案例以及当前工具的具体分类，输出则是 JSON 格式的一组子任务。每个子任务都是一个独立的任务单元，包括“子任务名称”、“子任务描述”、“子任务类别”等重要信息。每个子任务都可以看作一个完整的任务进行执行。

首先，任务解耦模块的核心功能是将用户提出的复杂或模糊需求拆解为可执行的子任务。许多用户在表达需求时，往往由于信息不明确或需求过于笼统，导致系统难以直接响应。例如，用户可能提出多个相关或不相关的要求，或者在一个指令中混合了不同领域的子任务。在这种情况下，大语言模型通过对用户输入的语义分析，将任务按照逻辑和类别进行解耦。每一个子任务将独立处理，从而避免因任务过于复杂而导致系统误解或错误执行。

总的来说，基于大语言模型的任务解耦模块通过解耦复杂任务、改写用户需求，并动态调整任务规划，使得该系统在执行复杂任务时更加灵活、精准且高效。

4.3.2 任务编排模块

为了更好地利用我们构建的工具图谱，并挖掘隐藏节点关系中的知识，我们开发了一个基于深度优先遍历的寻路算法。与“思维链”（Chain-of-Thought）或 ReACT 方法相比，该算法的优点在于该方法在图谱上进行可回溯的动态选路，能够防止错误传播的问题，并能够对整个工具空间进行更全面的探索。在动态选择 API 工具路径的同时，我们会维护一个“短期记忆”，即当前的路径和每一步路径选择（前进/回溯）的理由。同时，为了利用历史经验知识，我们将历史上类似的任务的正确工具路径作为提示词提供给模型，以辅助模型进行路径选择。最后，在该算法能够通过“自我反思

机制”来对路径进行判断和错误诊断，这一机制进一步提升了路径选择的准确率。

4.3.2.1　基于深度优先遍历的动态搜索算法

图 4.2为本文提出的基于深度优先的寻路算法。

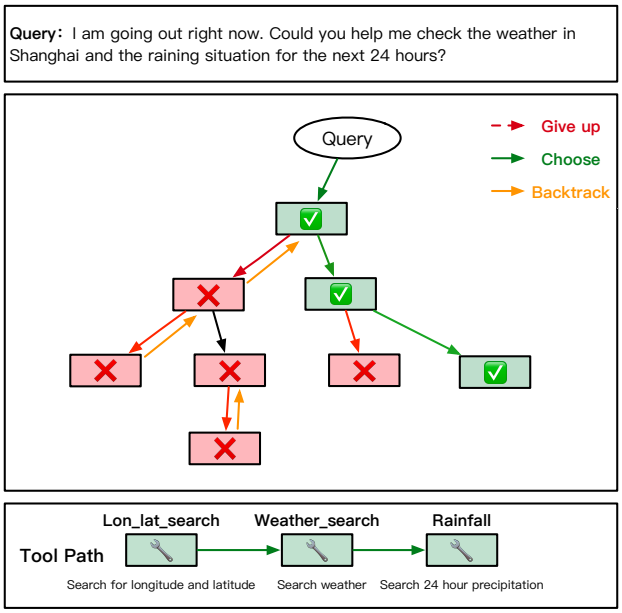

图 4.2　基于深度优先的寻路算法
Figure 4.2　Dynamic DFS Search on Tool Graph

首先，由于图谱上的节点数量众多，选择初始节点是算法中非常关键的一步。在选择初始节点时，我们训练了一个基于语义相似度的 API 召回器，能够根据用户的需求语句得到一组在语义上最相似的工具作为初始工具的候选集。然后我们从中选择一个节点并开始在图上进行深度优先搜索遍历，让模型在图上不断遍历其他节点。

具体算法逻辑如下：对于每个选择的节点，我们将获取当前节点的邻居节点及其权值，并提供给大模型智能体，要求大语言模型选择下一个工具节点或回溯。对于邻居节点候选集的筛选，我们会首先获取到图中所有邻居节点，然后我们根据当前节点与该邻居节点的工具转移边权进行排序，选择边权最大的 K 个节点作为候选邻居节点。同时，我们会获取每个邻居节点的点权，同样作为额外信息提供给大语言模型，作为工具可用性的参考。如果邻居节点不足 K 个，我们将再次调用 API 召回器，以补全 K 个候选节点的选项。

该算法会根据当前状态继续迭代选择下一个工具节点或发起回溯，直到模型决定结束选择或放弃该任务。

不管模型选择下一节点还是选择回溯，我们都将把模型的理由和具体操作加入短期记忆中，将回溯步骤添加到短期记忆的原因是让模型记住在本次推理中之前采取的错误操作，避免后续重复选择不可行的工具。如果模型回溯到了初始节点，并且需要继续回溯，我们可以理解为基于当前的工具选择，该任务不可行，即放弃任务执行。

4.3.2.2　初始节点召回器

在上述算法中，除初始节点外的其他点都是根据当前节点的邻居节点及其权值来选择的。因此，初始节点的选择是选择正确路径的关键。

由于工具图中包含大量的工具，无法让大语言模型浏览所有工具信息并选择最合适的。因此在这里我们设计了基于语义相似度的 API 召回器，能够根据用户的需求语句召回一组在语义上最相似的工具作为初始工具的候选集。

通常 API 召回是通过向量模型和相似度算法来进行的，具体而言，我们将用户需求（查询）和每个 API 的信息都转化为向量表示，并通过计算查询向量与 API 向量之间的相似度来检索相关 API。这个过程首先使用预训练的语言模型将 API 的文本描述转化为向量，这些向量能够捕捉文本的语义信息。接着，通过计算查询向量与 API 向量之间的相似度，常用的相似度度量包括余弦相似度和欧氏距离，系统会召回与查询最相关的若干个 API，通常会设置一个阈值，以确保返回的结果在一定的相似度范围内。随后，对召回的 API 进行排序，通常按照相似度从高到低排列，以便优先展示最相关的结果。在某些情况下，可以结合额外的规则或业务逻辑进一步优化结果，例如排除某些不相关的 API 或添加领域特定的过滤条件。

然而，我们在实际应用中发现，针对一些模糊的用户需求，开源的向量模型在召回工具列表时往往会收到噪声的影响。

如图 4.3所示，在工具召回时，有很多噪声工具被召回，而真正对任务有用的工具排名靠后。

因此，为了解决上述问题，本文提出了基于通用向量模型进行微调，通过构造高质量的工具训练数据来将领域知识注入模型，从而提升模型在工具选择上的准确性。

在训练向量模型时，训练数据包含三个部分：正样本，负样本，查询语句。查询语句即为数据集中的“query”字段，对应用户输入的查询信息。正样本也可以直接使用数据集提供的参考 API 列表。

对于负样本的部分，如图 4.4，我们采取了两种不同的负样本构造方式：一种是简单负样本（Simple Negative）构造，另一种是困难负样本构造（Hard Negative）。

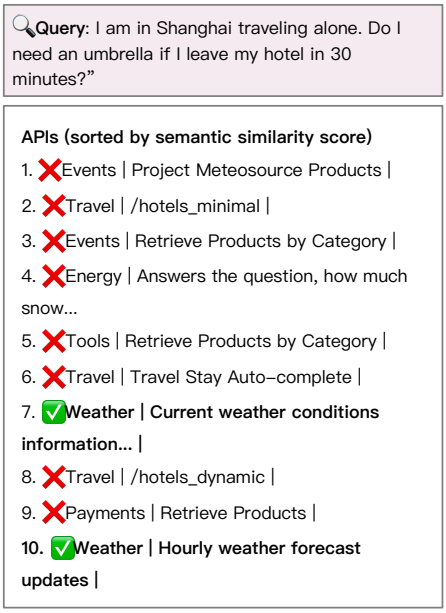

图 4.3　使用开源向量模型得到的 API 排序结果

Figure 4.3　API Ranking Using Open-source Embedding Models

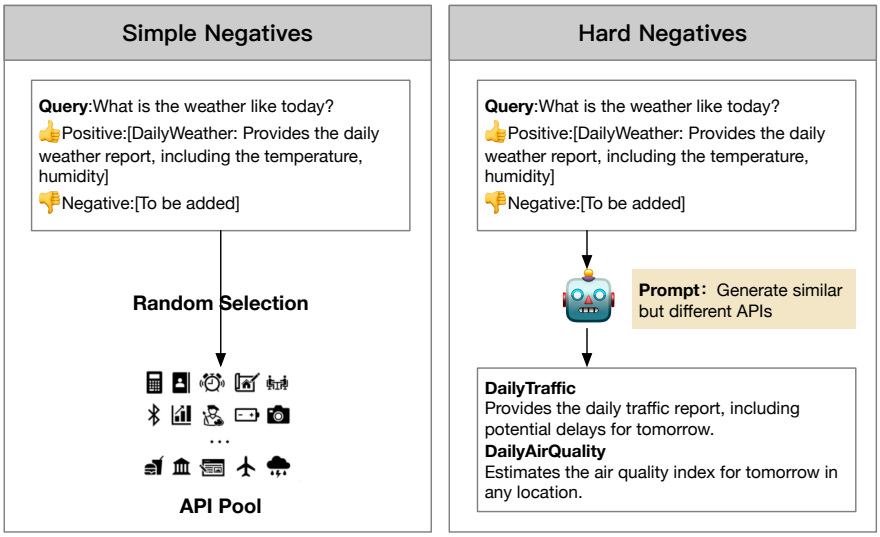

图 4.4　负样本构造的两种方式

Figure 4.4　Negative Sample Generation

简单负样本构造。对于简单负样本构造，我们直接选择不同类别的 K 个工具作为负样本。在简单负样本构造中，不同类别的 API 之间的功能上一般有区别，因此简单负样本构造能够保证负样本与正样本之间在工具上的差异。

困难负样本构造。在工具选择、调用场景，开源的通用向量模型难以区分工具之间细微的语义区别。困难负样本构造的目的是帮助模型更好地区分工具之间细微的语义区别，帮助更好地应对噪声。我们选择采用 GPT-3.5 完成困难负样本的构造。由于原工具数据量众多，从中筛选语义表述类似、但是功能上有区别的工具样本费时费力。因此我们采取了直接用大语言模型生成工具描述作为负样本的策略。

具体策略如下：首先，我们采样一批 <query, 推荐工具集 > 的数据，然后我们遍历其中的推荐工具集的工具，通过提示词将用户需求和工具描述提供给大语言模型，要求模型生成类似但是功能上无法满足用户需求的工具描述。通过这样遍历工具描述生成负样本，可以生成一组高质量的困难负样本供模型学习。

4.3.2.3　自我反思机制

自我反思机制的主要目的是对生成的工具调用路径进行反思，以提升整体的准确率。

自我反思机制的触发时间点有三个：1. 动态寻路算法找到路径并主动结束路径 2. 动态寻路算法超过了最大迭代次数并终止或放弃寻路

具体而言，自我反思模块的输入是我们当前的工具调用路径和用户需求，根据 API 工具召回器召回的初始节点，以及我们撰写的反思格式说明。输出包括以下几部分：1. 成功/部分失败/完全失败的等级 2. 若部分失败，从哪一个工具节点开始为首个错误节点 3. 若全部失败，有哪些可以删除的噪声工具初始节点

评价为“成功”的路径，我们直接进入 API 调用模块进行调用。对于评价为“失败”的路径，我们会根据失败的等级选择进行不同的重新寻找工具调用路径：第一种是从中间步骤开始重新寻找路径，另一种是从头开始重新寻找路径。

1. **从中间步骤继续**: 在任务未完成的情况下，我们将在短期记忆中记录寻路过程中的每一步的选择节点和理由。当路径被标记为“放弃”或被评判器认定为“失败”时，我们会重新激活该路径上的智能体，并将识别到的失败原因重新纳入历史上下文。评判器在判定“失败”时，通常会标记出它认为的第一个出现错误的节点。在重新激活智能体并进行寻路时，我们将从该节点继续，而不是从头开始。这种从中间步骤继续的策略不仅能够加快寻路速度，减少大语言模型的调用次数，还能充分利用先前成功调用的经验，从而提升决策的准确性。

2. **重新寻路**: 在自我反思模块认为路径“完全失败”时，我们需要从头开始重新生成整条路径。评判器会识别路径初始节点中与用户查询无关的工具名称作为反思的一部分。为了提高系统的整体效率，我们会首先从初始工具节点中移除这些无关的工具，避免大模型受到这些噪声的影响，从而选择无关的工具进行调用，导致后续调用出错。通过这一清理过程，我们能够有效减少噪声工具的影响，确保后续搜索的准确性。

   接下来，我们将会在经过噪声清理的工具组中重新开始选择下一节点并组成路径。这种从头开始的自我反思允许算法在一个更加简洁与优化的初始条件下进行搜索，从而提升工具调用路径的质量与响应速度。

该自我反思机制可以反复应用，直至满足终止条件为止。这种持续的反思过程确保了对问题的逐步优化，有助于形成更加有效的调用路径。

综上所述，这两种反思策略——从中间步骤继续优化和从头开始的寻路——的结合使用，能够在处理用户需求未满足的情况下，提供更高的灵活性与效率。通过不断的反思与优化，系统将逐步提升其在动态环境中的适应能力，确保用户体验的持续改进。

4.3.2.4　工具调用路径长短期记忆框架

本小节提出了关于增强模型规划和推理准确性的长短期记忆框架。该框架主要分为短期记忆和长期记忆两个部分。短期记忆部分指的是当模型在图上动态推理时存储的每个步骤的记录，主要聚焦于以什么数据格式来存储推理步骤，以及记录哪些有用的记忆信息：在图上前进、回溯还有大模型的思考过程等。长期记忆部分主要是将历史的工具调用路径存储在数据库中，通过相似度搜索算法加入到工具选择器的提示词中，以增强其推理和规划能力。

短期记忆

短期记忆指的是模型在图上进行不断推理时保存的一些状态信息，这些信息包括：用户的任务、当前遍历到的节点、历史遍历的节点、调用路径信息等。在遍历的过程中，我们会动态地更新和维护短期记忆存储的内容，来辅助模型进行推理和规划。在短期记忆中，我们一般将全部的信息存储在内存中，然后每次直接构建提示词添加到大语言模型的上下文中，让模型能够感知到当前的状态和环境。

长期记忆

长期记忆与短期记忆相对，是固定存储在数据库中的信息，且会随着调用次数的增加不断积累。它记录最终形成的工具调用链和结果，每次推理时，系统都会从长期

记忆库中搜索，将搜到的相似需求的工具调用路径提供给工具选择器，利用历史经验辅助大语言模型的推理。

长期记忆模块基于检索增强生成（Retrieval-Augmented Generation, RAG）逻辑，通过将历史上成功的 < 用户需求，工具调用路径 > 转为向量存储。新需求通过向量模型转为嵌入向量，并与数据库中的向量进行相似度计算。系统会根据相似度排序，检索出与当前需求最相似的 K 个历史需求及其工具调用路径。最终，这些 < 用户需求，工具调用路径 > 的二元组会被加入大语言模型的上下文中，辅助推理与规划。

4.3.3 工具调用模块

4.3.3.1 整体逻辑

工具调用模块的主要功能是执行规划好的 API 调用路径，并将得到的结果返回给系统，从而为后续的推理和规划提供支持，最终生成符合用户需求的答案。

该模块的整体逻辑可以分为两个主要部分：工具调用和工具响应解析。具体而言，我们首先根据工具的描述信息和用户需求生成 API 调用的参数，然后通过代码生成请求体并通过 API 调用接口将请求体发送给目标 API 工具，以获取其响应。获得响应后，理论上我们可以直接将所有工具的调用结果直接提供给总结器，要求模型根据所有的 API 响应输出最终结果。

但由于每个响应的长度参差不齐，对于一些响应长度较长的 API 工具，可能单个 API 的响应数据就超出了大语言模型的上下文限制。因此，我们添加了一个基于大语言模型的响应压缩模块，通过对响应的压缩，将 API 响应的字符数控制在 1024 个字符以内，以缓解大语言模型的上下文限制。

4.3.3.2 工具参数生成

工具调用模块的主要工作就是生成 API 工具的请求体。在请求体的生成过程中，我们首先将与工具调用相关的内容提供给大语言模型智能体，要求其生成指定的 JSON 格式调用输入，包括但不限于工具调用的 URL、所需输入参数及工具的登录验证信息等。接着，我们对生成的 JSON 数据中的参数信息进行验证，并确定请求体中各个参数的结构和格式。

首先，我们会确认生成的参数 URL 与工具库中的 URL 是否匹配。其次，参数的正确性对 API 工具调用的成功至关重要，因此在执行调用工具代码之前，我们会对参数进行严格的校验。在这一步中，我们验证具体的参数数量、类型和名称，并对这些部分进行精确匹配。如果缺少必要的参数或参数类型不匹配，我们将返回固定的错

误信息，并要求参数解析器重新生成参数。这一过程会一直重复，直到获取正确的参数或超过最大尝试次数后放弃该 API 调用。

在成功验证 URL 和参数后，我们将调用固定的函数来执行 API 请求，并获取 JSON 格式的响应，随后将其提供给响应解析模块。

4.3.3.3　工具响应解析模块

生成请求体后，我们将其通过 API 调用接口发送给目标 API 工具，以获取相应的响应数据。响应数据通常是以 JSON 格式返回的，这其中可能包含大量的信息，而这些信息并不总是直接相关。在我们的实际实验中，我们通过分析发现许多 API 返回内容包含大量冗余信息，导致其长度过长，无法将调用结果输入到大语言模型中，直接使用大语言模型从中提取重要信息较为困难。因此，我们对 API 的响应结果进行了压缩。该模块的目标是在尽可能多地保留关键信息的同时减少 API 响应的长度，便于放入大语言模型的上下文中。

由于每个 API 的响应格式不是固定的，无法确定每个字段应该舍弃还是保留，因此我们采用大语言模型来分析示例响应，仅保留与用户需求相关的字段，以减小响应长度。

如图 4.5，响应压缩的逻辑如下所示：

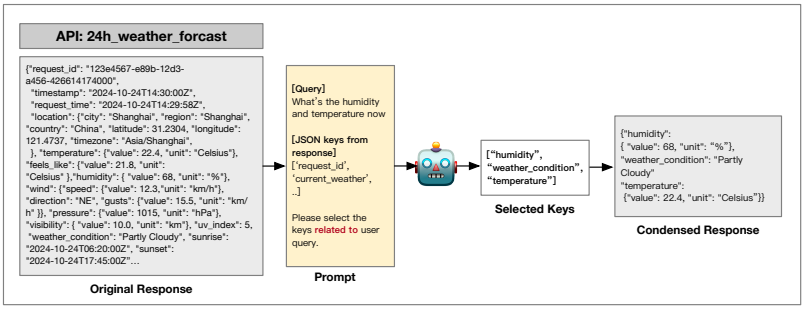

图 4.5　响应压缩模块
Figure 4.5　Compression of API responses

1. **工具文档信息**：所有 API 工具均来自 ToolBench 等开源数据集，这些数据集同时包含每个 API 的详细描述信息和 API 响应示例。因此，我们可以将工具名称、工具描述、API 名称、API 描述、参数及 API 响应示例的内容以文本形式提供给大型语言模型。这部分构成了压缩模块的基本提示词。
2. **详细规则指令**：我们要求大型语言模型仔细阅读 API 的功能描述，并保留与功能描述最相关的信息，诸如 API 版本、调用时间或无效信息等可以被舍弃。

3. **上下文学习示例**：我们使用了三个上下文学习示例，每个示例由一个原始 API 响应和对应的专家撰写的压缩响应组成。我们要求压缩器以自然语言文本格式输出所有需要保留的字段，然后通过正则表达式匹配得到一个保留字段的列表，并用固定的代码逻辑筛选出相应字段的返回内容。

在推理过程中，当 API 响应长度超过 1024 个字符时，我们会通过移除不重要的信息来压缩响应。如果压缩后的响应仍然超过 1024 个字符，则只保留压缩后的前 1024 个字符。这种方法能够有效地减少 API 响应长度，对 API 响应进行去噪。同时，也能够缓解大型语言模型有限的上下文窗口的问题，确保系统的正常调用。

通过分析我们在 ToolBench 中的 API 响应示例，其中一个 API 的平均响应字符串长度为 xxx 个字符。经过响应压缩后，最长的响应也不超过 1024 个字符。通过这种方式，我们有效降低了 xx% 的超出上下文的情况，平均每个 API 响应结果节约了 xx 个字符。

4.4 本章小结

第 5 章 实验分析

5.1 API 召回向量模型实验与评估

5.1.1 模型训练及超参数设置

根据第四章介绍的方法，我们采用了简单负样本构造和困难负样本构造两种方式，分别生成了 10000 条数据，用于微调模型。

在模型选择方面，我们选择了 BGE 系列模型，并最终确定 bge-m3 作为基座模型。BGE 模型在训练中使用了对比学习中的常用损失函数 **InfoNCE**，即信息噪声对比估计（Information Noise-Contrastive Estimation）损失。InfoNCE 通过最大化正样本与负样本之间的互信息来优化无监督表示学习，旨在增强模型的特征判别力。

在微调 bge-m3 过程中，我们将训练参数设置为：epoch 数量为 1，学习率为 3e-5。模型大小为 340M，在一台带有 V100 芯片的 GPU 上训练，最终获得了最优模型。

5.1.2 评估指标

我们使用召回率（Recall）和 NDCG 分数（包括 NDCG@3 和 NDCG@8）来衡量微调后的工具召回器的效果。

5.1.3 实验结果

【TODO】这里要区分原始的 simple-neg，hard-neg 之间的区别。还有 recall 和 ncdg 的区别。

表 5.1 API 工具召回实验结果

方法	Recall	NDCG@3 (I1)	NDCG@3 (I2)	NDCG@3 (I3)	NDCG@8 (I1)	NDCG
BCE	41.5	28.0	32.7	21.9	33.8	37
BGE-Small	53.4	36.1	38.4	27.6	43.2	48
ToolBench's API Retriever	57.8	39.2	42.9	33.8	47.0	51
Ours	**71.3**	**55.0**	**66.2**	**60.5**	**61.8**	**73**

5.1.4 实验分析

表 5.1 显示了不同方法在召回率（Recall）、NDCG@3 和 NDCG@8 指标上的表现。分析如下：

- **召回率 (Recall) 的提升**：我们的模型在召回率上达到了 71.3%，相比 BGE-Small 提升了 17.9 个百分点。ToolBench 的 API 检索器召回率为 57.8%，相较于本方法低 13.5 个百分点。这表明我们的方法能够识别出更多的相关工具，显著提高了召回覆盖率。
- **NDCG@3 的提升**：在更关注前 3 个排序结果的 NDCG@3 指标上，本方法在 I1、I2、I3 三类任务中分别达到了 55.0、66.2 和 60.5，相比 ToolBench 的 API 检索器平均提升了 20 个百分点。这表明我们的模型在前几个检索结果中的相关性排序更为精准，能够在关键的前几项中有效地呈现高相关性工具。
- **NDCG@8 的提升**：在关注前 8 个排序结果的 NDCG@8 指标上，本方法在 I1、I2、I3 三类任务中分别达到了 61.8、73.1 和 68.4，相比于 ToolBench 的 API 检索器平均提升了 19.1 个百分点。NDCG@8 的提升表明，本方法在扩大检索范围时仍能保持高质量的相关性排序，使得用户在更大范围内获得较为可靠的工具结果。
- **任务难度对模型表现的影响**：随着任务难度从 I1（单工具任务）增加到 I3（跨类别任务），所有模型的 NDCG 值均有所下降。这反映了任务复杂性对检索模型的挑战性。然而，本方法在 I2 和 I3 上的表现仍优于其他基线，说明了在多工具或跨类别任务中的鲁棒性更强。
- **召回率与排序质量的平衡**：结合召回率和 NDCG 的结果分析，可以看到我们的模型在保证高召回率的同时，能够更有效地将高相关性的工具排在前列。这一平衡对于工具检索任务尤为重要，特别是在需要优先呈现最佳工具的情境下。相比于基线方法，我们的方法能够在前 3 和前 8 个结果中提供更高的 NDCG 值，这表明在扩大候选工具集范围时，本方法仍具有较强的排序优先级。

综上所述，实验结果表明，我们的方法在 API 工具召回任务中取得了显著的性能提升。特别是在复杂任务和多类别需求中表现突出，进一步验证了基于对比学习微调的 bge-m3 模型在多工具组合和跨类别场景中的适用性。

5.2　基于工具图谱与深度优先遍历的 API 编排与调用方法实验

5.2.1　测试集构造

数据集介绍：ToolBench[8]是一个公开的针对工具调用的数据集，其中包含了来自 49 个类别的 16464 个真实世界的 API 工具的推理调用数据。该数据集包括三个部分，三个子数据集的难度逐级上升：G1 数据集，其中目标任务所需的 API 都在同一

个工具组；G2 数据集，其中目标任务所需的 API 在同一个类别但是属于不同的工具组；G3 数据集，其中目标任务所需的 API 会跨越不同类别。为了测试各个难度等级上的能力，本工作从三个类别分别抽取了 350，350 和 300 条数据构建测试集。测试集一共涉及 18 个 category 的 358 个工具。

考虑到 RapidAPIHub 上的 API 的质量参差不齐，比如有一些 API 工具为废弃的，并且存在一大部分 API 工具为付费工具，这都可能会给测试过程引入不必要的噪声。

因此本工作首先筛选得到了一组覆盖各种类别的已知可用的高质量 API 工具，然后针对这些 API 工具，沿用 ToolBench 的方法构建了三个不同难度的测试数据集，作为该方法的测试数据。下面将会详细介绍数据集的构建过程。

5.2.1.1 高质量 API 工具集筛选

首先，我们需要定义什么是高质量的 API 工具。在我们的使用场景中，工具的可用性是首要考虑因素，因此必须确保筛选后的 API 工具都是可用的。此外，在工具选择模块中，我们使用 API 工具的名称和描述信息作为输入，供模型参考和选择。因此，API 名称的易读性和描述的丰富性也是筛选时的重要参考标准。

同时，在保证 API 质量的基础上，我们也希望尽可能覆盖更多的工具类别和工具集。因此，我们从每个不同类别的工具中进行采样，选择了共计 xx 个类别、xx 个工具集的 xx 个工具，作为筛选前的工具池。

基于上述规则，我们构建了一个工具筛选流程，并针对不同维度设置了相应的筛选机制。对于 API 工具的可用性，我们通过调用示例代码来测试每个工具的有效性。根据 API 的返回状态码、请求响应时间和响应内容，我们选择最合适的 API。在我们评估的 xx 个 API 工具中，有 xx 个 API 的响应状态码为错误码，且有 xx 个 API 未能在规定的时间内返回。经过可用性筛选后，我们从 xx 个工具中筛选得到了 xx 个可用工具。

对于 API 描述的丰富性和完整性，我们采用大语言模型标注的方法进行筛选。我们构建了包含详细指令和筛选标准的提示词，并提供了 few-shot 样例，供模型对每个 API 进行评估。为加快筛选速度并节约模型调用的字数，每次将 K 个 API 进行批量判断。模型将输出一个 JSON 格式的列表，包含对每个 API 的保留或丢弃的判断。

经过第二轮筛选后，最终剩下的高质量 API 工具共有 xx 个。

画表格，介绍每个不同部分有哪些 API 种类。

5.2.1.2　工具调用数据集构造

为了覆盖不同难度和复杂度的用户需求，我们参考 ToolBench 中的分类方法，选择了三个不同难度的任务类别：单工具任务、多工具集任务和多类别任务。

1. **单工具任务**

   该任务仅涉及一个工具，用户需求仅包含一个工具的调用。这是工具调用中最简单的情况，通常用于测试大语言模型在处理基本指令时的能力。在数据生成过程中，我们直接随机采样一些 API，并引导大语言模型生成与这些 API 相关的用户需求。这种方法不仅能够快速生成数据，还能确保指令的有效性和准确性，适用于初学者或对工具调用不太熟悉的用户。

2. **多工具集任务**

   该任务涉及多个工具集，用户需求需要调用多个工具集中的多个工具。这种任务要求大语言模型具备更高的灵活性和综合能力，能够理解不同工具之间的功能关系。在实现时，我们随机采样来自不同工具集的工具，并将其提供给大语言模型，让其生成用户需求。为了确保生成的需求合理，我们特别考虑了工具组合的有效性。对于那些功能上明显重复或无法自然组合在一起的工具 API，大语言模型将直接放弃生成不合逻辑的用户需求，并重新采样一组更合理的 API。这种方法有效地增强了模型在实际应用中的适应性，帮助生成更符合真实场景的用户需求。

3. **多类别任务**

   该任务涉及多个类别，用户需求需要调用多个类别的多个工具。这是对大语言模型综合能力的进一步挑战，因为不同类别的工具可能具有不同的功能和用途。在实现过程中，我们同样随机采样来自不同类别的工具，并将其提供给大语言模型，促使其生成多样化的用户需求。这种多类别的设计不仅提高了数据的复杂性，还增强了模型在处理多元化需求时的能力，使其更接近于真实世界的使用场景。

通过上述的方法，我们构建了一个共 1000 条数据的测试集，其中单工具、多工具集和多类别任务分别占 350、350 和 300 条。这种结构化的测试集设计使得我们能够全面评估大语言模型在处理不同复杂度的用户需求时的表现，进而优化模型的生成能力和适应性。经过人工的评估，这种方法具有较高的多样性，能够覆盖到大部分的实际场景。

5.2.1.3 基于工具图的测试数据构造

在测试数据构造过程中，我们参考了 TaskBench 的子图采样和反向指令生成方法：首先在工具图中采样一个子图，然后通过大型语言模型（LLM）将采样得到的子图转换为自然语言用户指令，从而构建测试数据集。

具体而言，我们从工具图中抽取子图，并保留抽取工具在原图中的连接关系，以表示工具之间的依赖性。我们将得到的工具子图分为两类：单个工具节点和工具链。

单个工具节点代表独立的工具调用，适用于单个工具即可完成的简单任务。工具链表示按顺序调用工具，需要多个工具依次执行，以完成较为复杂的任务。

通过上述两种子图抽样方式，我们可以模拟现实中的工具调用模式，以满足不同用户指令的需求。

我们将工具子图表示为 $G_s = \{T_s, E_s\}$，其中 $T_s = \{t_{s1}, t_{s2}, \ldots, t_{sk}\}$ 并且 $k < n$，$E_s = \{(t_{sa}, t_{sb})\}$，其中 t_{sa} 和 t_{sb} 属于 T_s。工具子图的抽样过程描述如下：

$$\text{Sample}(G, \text{type}, \text{n}) \rightarrow G_s$$

其中，type 指定抽样模式（如单节点、工具链），n 表示工具数量（范围设定为 $\{1, 2, \ldots, 5\}$）。这些因素决定了用户指令中工具子图的拓扑结构特性和节点数规模。

接下来，基于采样得到的子图 G_s，我们使用 GPT-3.5 等大型语言模型（LLM）生成用户指令。此过程称为反向指令生成（BACK-INSTRUCT），用于将采样得到的工具子图转换为自然语言用户指令。具体而言，给定一个抽样得到的子图 G_s，我们定义反向指令生成过程如下，以使 LLMs 能够生成相应的指令：

$$\text{BackInstruct1}(G_s = (T_s, E_s)) \rightarrow \text{Instruction}.$$

在此过程中，采样得到的子图 G_s 用于指导 LLM 生成涵盖相关子任务及其依赖关系的用户请求。该策略保证了生成数据的复杂性和质量。

按照上述数据构造策略，我们共生成了 1000 条数据作为测试集，其中包括 300 条单工具任务和 700 条多工具任务。该测试集共覆盖 846 个工具，多工具任务的平均工具节点数为 3.4，表明此数据集在工具调用任务上具有一定的复杂性。

5.2.2 评估指标

由于工具的多样性，对于同一个用户需求可以有多种工具调用路径。因此，我们无法事先对每个测试的输入标注单一的解决路径标准答案。由于人工评价较为费时

费力，本文基于[18]中的评估器构建了类似的评估体系，包含以下两个指标。我们的评估器使用的是目前能力最强的模型之一 GPT-4，温度系数设置为 0。（todo）

- **通过率（Pass Rate）**

  通过率是计算在有限的工具执行步骤内完成了需求的比例。该指标衡量了系统工具调用最基本的执行能力。通过率的公式如下：

$$PR = \frac{\#(\text{Solved})}{\#(\text{Solved}) + \#(\text{Unsolved})}. \tag{5.1}$$

- **胜率（Win Rate）**

  胜率是评价两条针对同一需求生成的路径的偏好。在模型判断胜率的评估器的提示词中，我们预先定义了一组标准，其中包括：探索性、真实性、工具个数。胜率的公式如下：

$$WR = \frac{\#(\text{Won})}{\#(\text{Won}) + \#(\text{Lost}) + \#(\text{Tie})}. \tag{5.2}$$

同时，为了验证评估器与人类标注者的标注一致性，我们人工标注了 100 条通过率和 100 条胜率的数据。经过这 200 条数据，我们发现标注器在通过率上与人工标注的一致性达到了 xxx（todo），在胜率上该数字达到了 xxx（todo），这表明该基于大语言模型的标注器与人工标注的标准基本吻合。

5.2.3　基准线

为了对比本研究提出的基于 Agent 与知识图谱的任务编排与执行方法的效果，本文选用下列方法作为实验的基准方法。

- **基本提示方法**。基本提示方法即在大语言模型中直接输入所有候选 API 的信息，然后要求大语言模型输出需要调用的 API 的名称和参数等。
- **思维链方法**[Wang2023]。思维链方式在提示词中加入了”Let’s think step by step” 的提示信息，引导大语言模型能够进行按步骤的推理。
- **ReACT 方法**[Yao2023]。ReACT 方法通过让大语言模型不断生成 Thought 和 Action，然后将外部的环境反馈也纳入大语言模型的上下文，让模型能够更好地进行规划。

关于大语言模型的选择，我们选择了 Qwen2.5-7b 和 GPT-3.5 作为基础模型，这两个模型都具有中英文双语能力，且能够对比开源模型和能力更强的闭源模型在工具能力上的区别。

5.2.4 实验结果

实验在构建的测试集上进行，评估了本研究方法与不同基线方法在通过率 (Pass Rate) 和胜率 (Win Rate) 指标上的表现。实验结果如表 5.2 所示。

表 5.2 不同方法在测试集上的实验结果比较。

方法	通过率 (Pass Rate)		胜率 (Win Rate)	
	G1	G2	G1	G2
基本提示方法	65.3%	54.1%	60.8%	50.2%
思维链方法[Wang2023]	72.4%	61.9%	68.7%	58.3%
ReACT 方法[Yao2023]	78.1%	66.2%	72.5%	62.7%
本研究方法 (Ours)	**85.2%**	**74.3%**	**81.4%**	**69.5%**

结果分析 表 5.2 展示了各方法在不同难度任务 (G1 和 G2) 上的通过率和胜率。从结果可以看出，本研究方法在所有指标上都优于其他基线方法，具体分析如下：

- **通过率 (Pass Rate) 的提升**：在 G1 数据集上，本方法的通过率达到了 85.2%，相比于 ReACT 方法的 78.1%，提高了 7.1 个百分点；相比于思维链方法提高了 12.8 个百分点。在更具挑战性的 G2 数据集上，本方法的通过率为 74.3%，比 ReACT 方法高出 8.1 个百分点，比思维链方法高出 12.4 个百分点。这一提升说明，本方法在处理单工具任务（G1）和多工具任务（G2）方面的鲁棒性更强，尤其在复杂任务场景中表现更为突出。
- **胜率 (Win Rate) 的优势**：在 G1 数据集上，本方法的胜率为 81.4%，高于 ReACT 方法的 72.5%，提升了 8.9 个百分点。在 G2 数据集上，胜率为 69.5%，相比于 ReACT 方法的 62.7% 提高了 6.8 个百分点。该胜率的优势表明，本方法在同一需求上生成的路径更符合优先排序要求，即在工具选择和调用的合理性上有较强的表现。
- **不同难度任务的对比**：可以观察到，所有方法在 G2 数据集上的通过率和胜率都比 G1 低。这表明多工具任务在工具组合和顺序上更具挑战性，但本方法在 G2 上依然保持了较高的通过率和胜率，相较基线方法提升显著，说明其在复杂任务上的适应性更强。

5.2.5 错误分析

尽管本方法在测试中表现良好，但在部分任务上仍存在失败情况。通过分析失败案例，发现主要有以下三类问题：

1. **API 接口调用错误**：在失败的案例中，有约 30% 是由于 API 接口调用错误造成的。这类错误包括参数缺失、数据类型不匹配等。改进建议是引入自动参数校正机制，以减少因参数错误导致的调用失败。
2. **多工具依赖问题**：有约 25% 的失败案例源于多工具依赖问题。某些工具输出的结果不符合下一个工具的输入需求，导致后续调用失败。引入更加严谨的依赖关系建模或自动数据格式转换机制可以有效提升模型的成功率。
3. **工具不兼容性**：在多工具组合中，有 18% 的失败案例是由于工具功能重叠或调用时的不兼容性造成的。今后可以增加工具的兼容性检查机制，以避免此类错误。

5.2.6　消融实验

为了验证工具图谱和深度优先遍历在方法中的重要性，我们进行了消融实验，分别移除不同组件并观察其对模型性能的影响，实验结果如表 5.3 所示。

表 5.3　工具图谱的消融实验结果。

实验设置	通过率 (Pass Rate)	胜率 (Win Rate)	执行步骤 (Step Count)
完整模型（含工具图谱）	**85.2%**	**81.4%**	**3.1**
无工具图谱	74.8%	69.2%	4.2
无深度优先遍历	78.3%	72.5%	3.7
无依赖关系建模	80.1%	76.1%	3.4

消融实验结果分析　表 5.3 的结果显示了不同组件对模型性能的影响：

- **工具图谱的作用**：移除工具图谱后，通过率从 85.2% 降至 74.8%，胜率从 81.4% 降至 69.2%，执行步骤数从 3.1 增加到 4.2。这说明工具图谱在减少无效调用和提高执行效率方面具有显著作用。没有工具图谱的情况下，模型需要更多的步骤来完成任务，且选择路径的合理性下降。
- **深度优先遍历的作用**：当移除深度优先遍历策略时，通过率从 85.2% 降至 78.3%，胜率从 81.4% 降至 72.5%。这表明深度优先遍历可以有效地优化工具调用的顺序，减少无效步骤，帮助模型在较短路径内完成任务。
- **依赖关系建模的影响**：移除依赖关系建模后，通过率从 85.2% 降至 80.1%，胜率从 81.4% 降至 76.1%。依赖关系建模的缺失会导致工具调用的前后顺序不合理，从而影响任务的成功率和路径选择的合理性。说明依赖关系建模在工具之间的协作上发挥了重要作用。

小结 消融实验结果表明，工具图谱和深度优先遍历策略在本方法中具有关键作用。工具图谱可以提供清晰的工具调用路径和依赖关系，有效减少执行步骤和提升任务成功率；深度优先遍历策略则能帮助模型优先完成必要的调用步骤，优化了执行效率。因此，结合工具图谱和深度优先遍历的方式对于复杂任务的高效解决至关重要。

5.3 本章小结

本章主要介绍了。

第 6 章 系统设计与实现

本章设计并实现了一个具有交互性和用户友好性的基于知识图谱和大语言模型智能体机制的 API 编排和调用系统。该系统集成了不同模型的调用接口，以及知识图谱查询的功能。该系统提供了一个使用门槛低、用户友好的界面，允许用户通过自然语言的方式与该系统进行交互和提问，系统会根据用户的需求编排并调用所需的 API，并进行回答。

难点：

界面设计：友好交互、展示图谱部分、添加 api 部分

前后端框架选什么

前：streamlit 后：fastapi

模型管理（不同模型，超参数，上下线）如何交互容错 restapi 格式如何加速

数据管理 neo4j qdrant 用户数据记忆数据

6.1 系统需求分析

我们采用面向对象的需求分析方法，绘制了如图 6.1所示的系统用例图。

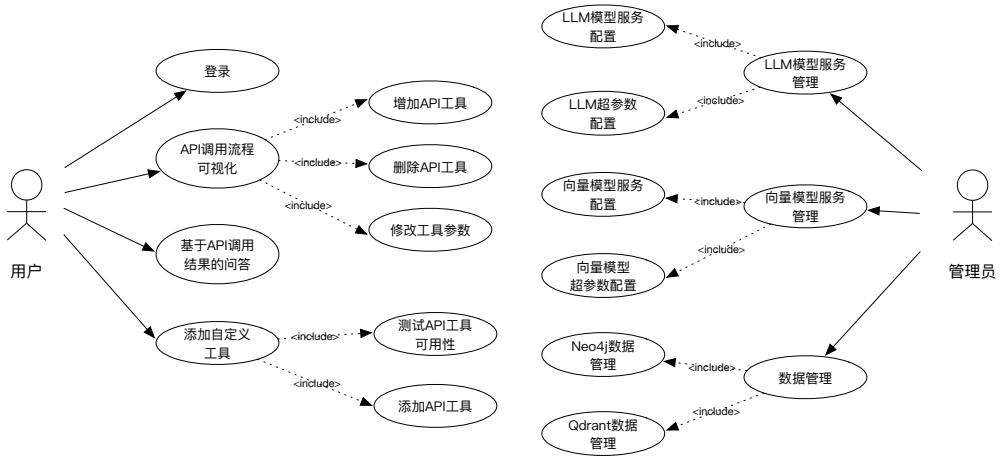

图 6.1 系统用例图
Figure 6.1 System Usecase Diagram

在用例图中，我们定义了两种角色：系统用户和管理人员。从需求出发，系统应该实现以下功能：

针对普通用户的功能：

1. 登录功能
2. API 调用流程可视化
3. 基于 API 调用结果的问答
4. 添加自定义工具

针对管理员的功能：

1. 大语言模型服务管理
2. 向量模型服务管理
3. 数据管理

在该系统中，基于大语言模型的智能体可以通过多轮对话的交互方式与用户进行交流，并通过解析用户的需求来提供对应的工具调用流程和根据工具调用结果得到的总结回复

本文的系统提供的主要功能是根据用户需求得到调用流程并执行，从而给用户提供有效的信息。针对该核心功能，我们针对不同用户群体设计了两种使用的模式：对于有计算机编程基础的用户，我们提供了开发者模式，即用户可以自己对工具执行流程进行编辑和调整，以确保更加符合其查询需求；对于一般用户，我们按照系统生成的流程进行执行，得到最终结果。除了该功能外，本系统还提供了自定义工具添加、工具模板库浏览等功能。

6.2　系统设计与架构

本文设计的基于知识图谱和智能体的 API 编排与调用系统，整体的系统框架设计如图 xx 所示。系统主要包含四层：数据管理层、API 编排层、API 调用层和 UI 层。

6.3　交互方案设计

图 6.3为该系统的交互设计图。

6.4　系统架构设计

图 6.3为该系统的系统软件架构设计图。

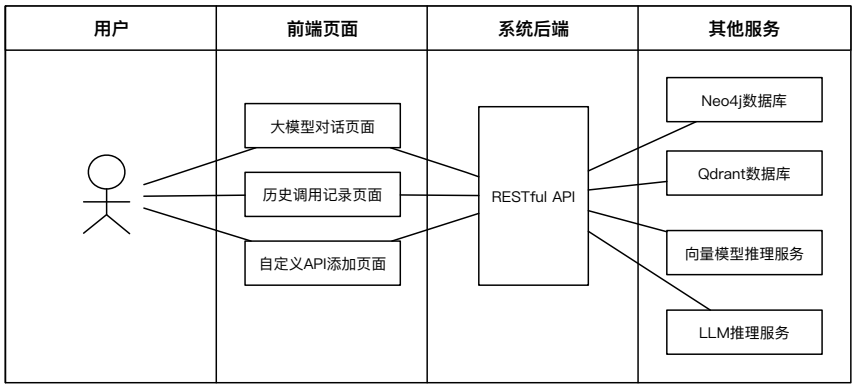

图 6.2 系统交互图
Figure 6.2 System Usecase Diagram

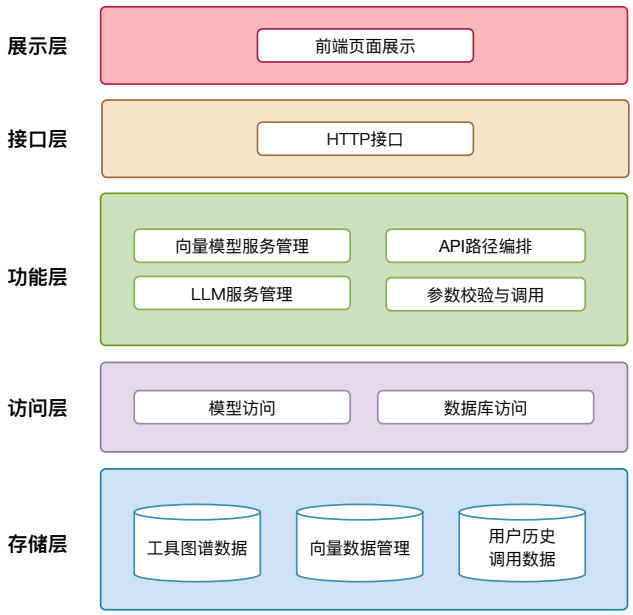

图 6.3 系统用例图
Figure 6.3 System Usecase Diagram

6.4.1　存储层

数据管理层主要负责下列内容的存储：

1. API 知识图谱的存储
2. 向量形式存储的 API 详细信息，这些预先计算并存储在向量数据库中，便于快速计算
3. 系统使用过程中的历史推理轨迹的存储，作为经验数据供后续参考

6.4.2　访问层

6.4.3　功能层

API 编排层主要实现本系统的动态 API 编排功能。该层主要包括以下几个部分：

1. **用户需求拆解模块**：通过大语言模型智能体机制，将用户的复杂、完整需求拆解为多个子需求，并对子需求逐一调用。
2. **API 推理轨迹检索模块**：根据子任务的任务描述，检索历史推理轨迹中相似任务，提供参考。
3. **API 知识图谱检索模块**：根据不同的检索模式，搜索 API 节点及相关信息作为返回。
4. **基于图谱的 DFS 动态编排算法模块**：在知识图谱上进行搜索和回溯，得到最终的调用路径。
5. **API 参数获取模块**: 首先，我们获取得到对应 API 的参数信息以及用户的具体需求，提供给大语言模型让模型给出 API 所需的输入参数。
6. **API 调用模块**: 在该部分，会使用 API 参数获取模块的参数作为 API 输入来调用工具 API，若执行成功则将 JSON 格式的调用结果传递到 API 调用结果总结模块。该部分设置了自动重试机制，若调用失败则自动重试 3 次，如果 3 次都失败则返回错误信息。
7. **API 调用结果总结模块**: 在执行并得到了 API 的调用结果后，将所有的 API 执行结果转化为自然语言的格式，将这些执行结果和用户原需求输入基于大语言模型的总结模块获取最终自然语言格式的总结内容。

6.4.4　接口层

接口层负责提供对各种功能的访问接口，采用的是 HTTP 接口的形式。系统后端通过提供 RESTful API 格式的 HTTP 接口来供前端调用和访问。模型服务与平台后端也是用过 HTTP 接口来实现高效、安全的通信。

6.4.5 展示层

UI 层是 web 前端页面，负责与用户直接交互，提供用户友好的界面。不同页面对应不同功能：

- **信息问答页面**：用户通过自然语言输入具体的需求，然后系统会进行 API 调用路径的编排并依次调用，最终将总结好的自然语言文本的结果也提供给用户。整体的交互方式与大模型多轮对话是相似的。
- **自定义 API 添加页面**：用户可以通过填写 API 的详细信息，如 API 的名称、API 的调用链接、API 的具体参数类型和参数描述信息等添加新的 API。添加新 API 后还需要经过测试，确保 API 调用的有效性再添加到 API 仓库中。
- **API 历史调用参考页面**：用户可以可视化的形式浏览历史 API 调用记录，并通过选择和配置参数的方式再次调用历史 API 调用链。

6.5 模块设计

图 xx 展示了系统详细的功能层次结构。本系统的详细功能主要包含：用户验证、数据管理、模型服务管理、API 工作流编排、API 调用问答、自定义 API 存储。

6.5.1 用户验证

用户验证模块用于实现面向用户的 API 调用历史记录和模板库构建。该模块包含用户注册和登录功能，允许用户通过注册获得个人账号并登录系统使用。

6.5.2 数据管理

数据管理模块用于管理系统中的各类数据，包括知识图谱数据、API 信息向量以及用户交互过程中的历史需求、API 编排和调用结果等。主要存储在 neo4j 和 Qdrant 向量数据库中。

6.5.3 模型服务管理

模型服务管理模块包括模型接口封装和模型超参数配置功能。封装不同模型的调用代码为统一接口，对系统其他部分隐藏模型的具体细节。超参数配置则管理模型调用时的参数，如温度系数、最大长度、top_k、top_p 等。

6.5.4 API 工作流编排

API 工作流编排模块包含以下三个子模块：

- **编排模式选择**：提供用户可配置的编排选项。
- **编排结果展示**：通过可编辑任务框展示 API 编排结果，任务框中展示 API 名称、描述、参数等信息。
- **API 流程自定义**：允许用户在生成的 API 流程基础上，进行增加、删除或修改，支持更灵活的 API 调用。

6.5.5　API 调用问答

API 调用问答模块通过流式输出和多轮交互，提升用户体验。流式输出模块增强用户等待结果时的体验，多轮交互模块保存上下文中的 API 调用结果，支持进一步提问。

6.5.6　自定义 API 存储

自定义 API 存储模块通过 API 添加和测试模块，支持用户将自定义 API 添加到系统中，提高系统扩展性和可用性。

6.6　系统实现

本节将会介绍系统的实现方法。首先是技术选型方面，考虑到与大语言模型、深度学习有关的代码都采用 Python 编写，我们也采用了 Python 技术栈。我们采用了前后端分离的方式，选择了 FastAPI 作为后端框架，前后端之间通过 HTTP 接口进行网络通信。在前端页面部分，我们选用了目前大语言模型 Chatbot 有关最流行的前端框架之一，Streamlit 前端框架来实现。

6.7　本章小结

第 7 章 总结与展望

本章共包括两个部分，第一个部分是对全文工作内容进行总结和回顾，第二个部分就是对本工作的不足和局限性进行探讨，并对未来可能的研究探索方向进行探讨。

7.1 工作总结

本文从 xx。

本文工作主要包含以下三个部分的内容。

7.2 未来展望

本文围绕着如何构建有效的 API 工具图谱、并将大量 API 工具集成到大语言模型中进行了一系列的探索，然而本文工作仍面临着许多不足之处和可以持续探索的空间。以下几点是未来可以持续探索和改进的方向。

1. 本文中提出的基于工具轨迹调用数据构建工具图谱的方法，在工具图谱的构建过程中，我们只考虑了工具之间的跳转关系，而没有考虑到工具之间的替换关系。然而在现实工具调用场景中，当一个工具不稳定或者实效时，用户可能需要选择其他在功能上有替换关系的工具进行代替。如何在工具图谱中考虑更多的工具之间的关系，并挖掘更丰富的工具关系，是未来值得探索的方向。

2. 本文中并未对大语言模型进行微调，而是直接使用了训练好的开源模型或 GPT-3.5 等闭源模型。但是，大语言模型工具集成的一个重要研究方向就是如何通过微调让模型获得更好的工具调用效果。现在有许多工作聚焦于微调规模较小的大语言模型，以提升其对工具任务的理解能力和工具方面的规划、推理能力。经过大规模语料预训练的模型中具有许多工具调用的知识，在未来如何通过有监督的微调来更好挖掘大模型在工具方面的潜力，并有效提升工具调用的效果，也是下一步需要研究的方向。

3. 为了提升系统的易用性和用户体验，我们可以进一步优化系统的界面设计和交互设计，使其更加直观和用户友好。同时，本系统仅在少量用户的情况下进行了测试，未来可以对该系统的性能和稳定性进行持续改进，以确保在大规模用户同时使用的情况下依然能够保持高效和稳定地提供服务。

参考文献

[1] RUAN J, CHEN Y, ZHANG B, et al. TPTU: Large Language Model-based AI Agents for Task Planning and Tool Usage[J]. 2023.

[2] SHEN Y, SONG K, TAN X, et al. HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face[J/OL]. 2023. https://arxiv.org/abs/2303.17580v4.

[3] SONG Y, XIONG W, ZHU D, et al. RestGPT: Connecting Large Language Models with Real-World RESTful APIs[EB/OL]. arXiv. 2023. http://arxiv.org/abs/2306.06624.

[4] MIAO N, TEH Y W, RAINFORTH T. SelfCheck: Using LLMs to Zero-Shot Check Their Own Step-by-Step Reasoning[J]. 2023.

[5] LIU Z, LAI Z, GAO Z, et al. Controlllm: Augment language models with tools by searching on graphs[J]. arXiv preprint arXiv:2310.17796, 2023.

[6] JONES K S. A statistical interpretation of term specificity and its application in retrieval[J]. Journal of documentation, 1972, 28: 11-21.

[7] ROBERTSON S, ZARAGOZA H, et al. The probabilistic relevance framework: BM25 and beyond [J]. Foundations and Trends® in Information Retrieval, 2009, 3: 333-389.

[8] QIN Y, LIANG S, YE Y, et al. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs[J]. 2023.

[9] WANG X, WEI J, SCHUURMANS D, et al. Self-Consistency Improves Chain of Thought Reasoning in Language Models[EB/OL]. arXiv. 2023. http://arxiv.org/abs/2203.11171.

[10] RAMAN S S, COHEN V, ROSEN E, et al. Planning with large language models via corrective re-prompting[C]//NeurIPS 2022 Foundation Models for Decision Making Workshop. 2022.

[11] YAO S, YU D, ZHAO J, et al. Tree of Thoughts: Deliberate Problem Solving with Large Language Models[J]. 2023.

[12] BESTA M, BLACH N, KUBICEK A, et al. Graph of Thoughts: Solving Elaborate Problems with Large Language Models[J/OL]. 2023. https://arxiv.org/abs/2308.09687v3.

[13] YAO S, ZHAO J, YU D, et al. ReAct: Synergizing Reasoning and Acting in Language Models [EB/OL]. arXiv. 2023. http://arxiv.org/abs/2210.03629.

[14] WANG G, XIE Y, JIANG Y, et al. Voyager: An Open-Ended Embodied Agent with Large Language Models[EB/OL]. arXiv. 2023. http://arxiv.org/abs/2305.16291.

[15] HUANG W, XIA F, XIAO T, et al. Inner monologue: Embodied reasoning through planning with language models[J]. arXiv preprint arXiv:2207.05608, 2022.

[16] LIU X, PENG Z, YI X, et al. ToolNet: Connecting large language models with massive tools via tool graph[J]. arXiv preprint arXiv:2403.00839, 2024.

[17] DU Y, WEI F, ZHANG H. Anytool: Self-reflective, hierarchical agents for large-scale api calls[J]. arXiv preprint arXiv:2402.04253, 2024.

[18] TANG Q, DENG Z, LIN H, et al. ToolAlpaca: Generalized Tool Learning for Language Models with 3000 Simulated Cases[J/OL]. 2023. https://arxiv.org/abs/2306.05301v2.

[19] MEKALA D, WESTON J, LANCHANTIN J, et al. TOOLVERIFIER: Generalization to New Tools via Self-Verification[J]. arXiv preprint arXiv:2402.14158, 2024.

[20] LUO L, LI Y F, HAFFARI G, et al. Reasoning on Graphs: Faithful and Interpretable Large Language Model Reasoning[EB/OL]. arXiv. 2023. http://arxiv.org/abs/2310.01061.

[21] SUN J, XU C, TANG L, et al. Think-on-graph: Deep and responsible reasoning of large language model with knowledge graph[J]. arXiv preprint arXiv:2307.07697, 2023.

[22] MA S, XU C, JIANG X, et al. Think-on-Graph 2.0: Deep and Interpretable Large Language Model Reasoning with Knowledge Graph-guided Retrieval[J]. arXiv preprint arXiv:2407.10805, 2024.

[23] JIANG X, XU C, SHEN Y, et al. On the Evolution of Knowledge Graphs: A Survey and Perspective [J]. 2023.

[24] 马展, 王岩, 王微微, 等. 基于多源信息融合的 API 知识图谱构建[EB/OL]. http://cnjournals.com/view_abstract.aspx?aid=CFC3B0096A6D80B2BB9723DA2B12C510&jid=D4F6864C950C88FFCE5B6C948A639E39&pcid=5B3AB970F71A803DEACDC0559115BFCF0A068CD97DD29835&yid=9475FABC7A03F4AB.

[25] LI H, LI S, SUN J, et al. Improving api caveats accessibility by mining api caveats knowledge graph [C]//2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2018: 183-193.

[26] LING C Y, ZOU Y Z, LIN Z Q, et al. Graph embedding based API graph search and recommendation[J]. Journal of Computer Science and Technology, 2019, 34: 993-1006.

[27] WANG X, LIU X, LIU J, et al. A novel knowledge graph embedding based API recommendation method for Mashup development[J]. World Wide Web, 2021, 24: 869-894.

[28] OpenAI. GPT-4 Technical Report[J/OL]. 2023. https://arxiv.org/abs/2303.08774v3.

[29] YANG A, XIAO B, WANG B, et al. Baichuan 2: Open Large-scale Language Models[J]. 2023.

[30] ZENG A, LIU M, LU R, et al. AgentTuning: Enabling Generalized Agent Abilities for LLMs [J/OL]. 2023. https://arxiv.org/abs/2310.12823v2.

[31] TOUVRON H, LAVRIL T, IZACARD G, et al. LLaMA: Open and Efficient Foundation Language Models[J/OL]. 2023. https://arxiv.org/abs/2302.13971v1.

[32] M. E N Z S. The Stanford Encyclopedia of Philosophy[Z]. 2019.

[33] XI Z, CHEN W, GUO X, et al. The Rise and Potential of Large Language Model Based Agents: A Survey[J]. 2023.

[34] WOOLDRIDGE M, JENNINGS N R. Intelligent agents: Theory and practice[J]. The knowledge engineering review, 1995, 10: 115-152.

[35] SUTTON R S, BARTO A G. Reinforcement learning: An introduction[M]. MIT press, 2018.

[36] WANG L, MA C, FENG X, et al. A Survey on Large Language Model based Autonomous Agents [J]. 2023.

[37] WENG L. LLM Powered Autonomous Agents[EB/OL]. 2023. https://lilianweng.github.io/posts /2023-06-23-agent/.

[38] GE Y, HUA W, MEI K, et al. Openagi: When llm meets domain experts[J]. Advances in Neural Information Processing Systems, 2024, 36.

[39] YE X, YAVUZ S, HASHIMOTO K, et al. RnG-KBQA: Generation Augmented Iterative Ranking for Knowledge Base Question Answering[J/OL]. Proceedings of the Annual Meeting of the Association for Computational Linguistics, 2021, 1: 6032-6043. https://arxiv.org/abs/2109.08678v2. DOI: 10.18653/v1/2022.acl-long.417.

[40] SHU Y, YU Z, LI Y, et al. TIARA: Multi-grained Retrieval for Robust Question Answering over Large Knowledge Bases[J/OL]. Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, 2022: 8108-8121. https://arxiv.org/abs/2210.12925 v1. DOI: 10.18653/v1/2022.emnlp-main.555.

[41] ALKHAMISSI B, LI M, CELIKYILMAZ A, et al. A Review on Language Models as Knowledge Bases[EB/OL]. arXiv. 2022. http://arxiv.org/abs/2204.06031.

[42] JI Z, LEE N, FRIESKE R, et al. Survey of hallucination in natural language generation[J]. ACM Computing Surveys, 2023, 55: 1-38.

[43] ZHANG Z, HAN X, LIU Z, et al. ERNIE: Enhanced Language Representation with Informative Entities[J]. 2019.

[44] LIN B Y, CHEN X, CHEN J, et al. KagNet: Knowledge-Aware Graph Networks for Commonsense Reasoning[J]. 2019.

[45] YASUNAGA M, BOSSELUT A, REN H, et al. Deep Bidirectional Language-Knowledge Graph Pretraining[J]. 2022.

[46] CHOUDHARY N, REDDY C K. Complex Logical Reasoning over Knowledge Graphs using Large Language Models[J]. 2023.

[47] LI M, ZHAO Y, YU B, et al. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs [EB/OL]. arXiv. 2023. http://arxiv.org/abs/2304.08244.

[48] LIU X, YU H, ZHANG H, et al. AgentBench: Evaluating LLMs as Agents[J]. 2023.

附录 A Maxwell Equations

选择二维情况，有如下的偏振矢量：

$$\mathbf{E} = E_z(r,\theta)\hat{\mathbf{z}}, \tag{A.1a}$$
$$\mathbf{H} = H_r(r,\theta)\hat{\mathbf{r}} + H_\theta(r,\theta)\hat{\boldsymbol{\theta}}. \tag{A.1b}$$

对上式求旋度：

$$\nabla\times\mathbf{E} = \frac{1}{r}\frac{\partial E_z}{\partial\theta}\hat{\mathbf{r}} - \frac{\partial E_z}{\partial r}\hat{\boldsymbol{\theta}}, \tag{A.2a}$$
$$\nabla\times\mathbf{H} = \left[\frac{1}{r}\frac{\partial}{\partial r}(rH_\theta) - \frac{1}{r}\frac{\partial H_r}{\partial\theta}\right]\hat{\mathbf{z}}. \tag{A.2b}$$

因为在柱坐标系下，$\overline{\overline{\mu}}$ 是对角的，所以 Maxwell 方程组中电场 $\mathbf{E}$ 的旋度：

$$\nabla\times\mathbf{E} = \mathrm{i}\omega\mathbf{B}, \tag{A.3a}$$
$$\frac{1}{r}\frac{\partial E_z}{\partial\theta}\hat{\mathbf{r}} - \frac{\partial E_z}{\partial r}\hat{\boldsymbol{\theta}} = \mathrm{i}\omega\mu_r H_r\hat{\mathbf{r}} + \mathrm{i}\omega\mu_\theta H_\theta\hat{\boldsymbol{\theta}}. \tag{A.3b}$$

所以 $\mathbf{H}$ 的各个分量可以写为：

$$H_r = \frac{1}{\mathrm{i}\omega\mu_r}\frac{1}{r}\frac{\partial E_z}{\partial\theta}, \tag{A.4a}$$
$$H_\theta = -\frac{1}{\mathrm{i}\omega\mu_\theta}\frac{\partial E_z}{\partial r}. \tag{A.4b}$$

同样地，在柱坐标系下，$\overline{\overline{\epsilon}}$ 是对角的，所以 Maxwell 方程组中磁场 $\mathbf{H}$ 的旋度：

$$\nabla\times\mathbf{H} = -\mathrm{i}\omega\mathbf{D}, \tag{A.5a}$$
$$\left[\frac{1}{r}\frac{\partial}{\partial r}(rH_\theta) - \frac{1}{r}\frac{\partial H_r}{\partial\theta}\right]\hat{\mathbf{z}} = -\mathrm{i}\omega\overline{\overline{\epsilon}}\mathbf{E} = -\mathrm{i}\omega\epsilon_z E_z\hat{\mathbf{z}}, \tag{A.5b}$$
$$\frac{1}{r}\frac{\partial}{\partial r}(rH_\theta) - \frac{1}{r}\frac{\partial H_r}{\partial\theta} = -\mathrm{i}\omega\epsilon_z E_z. \tag{A.5c}$$

由此我们可以得到关于 E_z 的波函数方程：

$$\frac{1}{\mu_\theta\epsilon_z}\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial E_z}{\partial r}\right) + \frac{1}{\mu_r\epsilon_z}\frac{1}{r^2}\frac{\partial^2 E_z}{\partial\theta^2} + \omega^2 E_z = 0. \tag{A.6}$$

附录 B 绘制流程图

图 B.1 是一张流程图示意。使用 tikz 环境，搭配四种预定义节点（`startstop`、`process`、`decision` 和 `io`），可以容易地绘制出流程图。

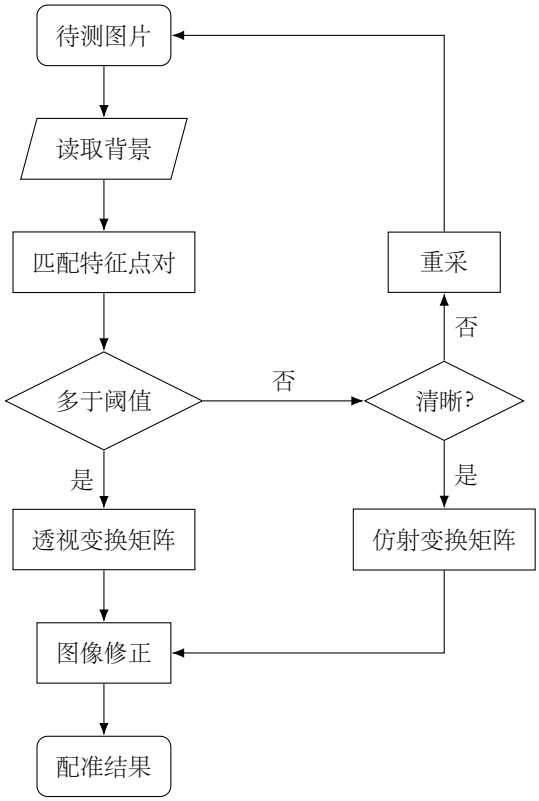

图 B.1 绘制流程图效果
Figure B.1 Flow chart

致 谢

感谢那位最先制作出博士学位论文 LaTeX 模板的物理系同学!
感谢 William Wang 同学对模板移植做出的贡献!
感谢 @weijianwen 学长开创性的工作!
感谢 @sjtug 对 0.10 及之后版本的开发和维护工作!
感谢所有为模板贡献过代码的同学们, 以及所有测试和使用模板的各位同学!
感谢 LaTeX 和 SJTUThesis，帮我节省了不少时间。

学术论文和科研成果目录

专利

[1] 第一发明人，“一种基于智能体与知识图谱的任务编排方法与系统”，专利申请号202411237239.0.

A SAMPLE DOCUMENT FOR LaTeX-BASED SJTU THESIS TEMPLATE

An imperial edict issued in 1896 by Emperor Guangxu, established Nanyang Public School in Shanghai. The normal school, school of foreign studies, middle school and a high school were established. Sheng Xuanhuai, the person responsible for proposing the idea to the emperor, became the first president and is regarded as the founder of the university.

During the 1930s, the university gained a reputation of nurturing top engineers. After the foundation of People's Republic, some faculties were transferred to other universities. A significant amount of its faculty were sent in 1956, by the national government, to Xi'an to help build up Xi'an Jiao Tong University in western China. Afterwards, the school was officially renamed Shanghai Jiao Tong University.

Since the reform and opening up policy in China, SJTU has taken the lead in management reform of institutions for higher education, regaining its vigor and vitality with an unprecedented momentum of growth. SJTU includes five beautiful campuses, Xuhui, Minhang, Luwan Qibao, and Fahua, taking up an area of about 3 225 833 m^2. A number of disciplines have been advancing towards the top echelon internationally, and a batch of burgeoning branches of learning have taken an important position domestically.

Today SJTU has 31 schools (departments), 63 undergraduate programs, 250 masters-degree programs, 203 Ph.D. programs, 28 post-doctorate programs, and 11 state key laboratories and national engineering research centers.

SJTU boasts a large number of famous scientists and professors, including 35 academics of the Academy of Sciences and Academy of Engineering, 95 accredited professors and chair professors of the "Cheung Kong Scholars Program" and more than 2000 professors and associate professors.

Its total enrollment of students amounts to 35 929, of which 1564 are international students. There are 16 802 undergraduates, and 17 563 masters and Ph.D. candidates. After more than a century of operation, Jiao Tong University has inherited the old tradition of "high starting points, solid foundation, strict requirements and extensive practice." Students from SJTU have won top prizes in various competitions, including ACM International Colle-

giate Programming Contest, International Mathematical Contest in Modeling and Electronics Design Contests. Famous alumni include Jiang Zemin, Lu Dingyi, Ding Guangen, Wang Daohan, Qian Xuesen, Wu Wenjun, Zou Taofen, Mao Yisheng, Cai Er, Huang Yanpei, Shao Lizi, Wang An and many more. More than 200 of the academics of the Chinese Academy of Sciences and Chinese Academy of Engineering are alumni of Jiao Tong University.