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 10 月 21 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

October 21st, 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。

知识图谱 blablabla。

该系统利用大语言模型的语义理解能力，整合外部数据，实时调用多种工具，满足用户对消费推荐、出行规划和信息查询的多样化需求。系统还具备动态调整能力，能够在用户行为和偏好发生变化时快速响应并优化推荐方案，确保系统在复杂的消费和出行场景下保持高效运作。

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。

这些问题制约了大语言模型在消费、出行等领域更大规模、更复杂场景中的广泛应用。

1.2　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：

- 如何将大语言模型与工具集成，实现工具调用和集成?
- 如何处理工具之间的依赖关系，实现工具的动态调整?

1.3　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. 研究了 API 过程数据的采集和 API 知识图谱的搭建。【需要补充】
2. 研究了基于知识图谱的 DFS 动态搜索算法。【需要补充】基于上述算法和机制，设计并构建了基于 Agent 与图谱的任务编排的具体流程。该流程以用户与系统对话的方式建立，可以通过自然语言输入用户的需求，并通过任务分解、图谱搜索和 API 编排等环节最终得到 API 调用链，并调用 API 得到结果。
3. 设计并实现了系统的各个功能模块，包括知识采集与图谱搭建、基于知识图谱的 DFS 动态搜索、基于语义相似度的长期记忆搜索、用户自定义 API 编排流程等功能，以及这些功能所对应的可视化系统界面。

1.4　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。

1.4.1 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。

1.4.1.1 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。

1.4.1.2　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。

1.4.2 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。

1.5　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来将轨迹中包含的工具知识放入图中，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。本章聚焦于如何搭建一个智能体流程框架来完成任务分解、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。

1.6 本章小结

本章概述了研究背景和本文的研究意义，介绍了国内外的研究现状与进展，以及本文的研究问题和具体内容。最后，介绍了论文的组织结构。

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，这些模型包含数十亿到百亿个参数，并在海量的预训练文本数据上进行训练。相比起预训练语言模型（PLMs），大语言模型具有更大的模型规模，并具有更强的语言理解能力和生成能力。更重要的是，随着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。

现有的大模型基本分为两种，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 工程技术

2.3.1 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。

2.3.2　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。

2.3.3　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。

2.3.4　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 基于工具调用轨迹的知识图谱构建

3.1 引言

本章介绍了一种基于工具调用轨迹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，比如：1. 如何对工具调用轨迹数据进行数据筛选，得到高质量的工具集以及工具调用轨迹？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？

基于上述问题，我们进行了以下研究：首先，我们提出了两种数据筛选策略，从 API 工具和 API 调用轨迹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，用于后续指导大语言模型进行选择。最后，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。

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，比如“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轨迹中的“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轨迹。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轨迹数据进行数据筛选、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：静态和动态图谱构建，两种构建都对图谱上的转换权值进行修改，两种方法互相辅助表示工具之间的转换关系。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。

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。关于图谱利用的部分会在下一章详细介绍。

3.3　数据收集

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 个 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。

【画图来展示 rapidAPI hub 的内容】

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 生成的真实调用轨迹和参考 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 生成的工具调用轨迹数据。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。

ToolBench 的工具调用轨迹数据为一个树状遍历结构，如下所示。

【画图来展示轨迹数据结构】

API-Bank 包括 xxx。

数据格式如下:

ToolAlpaca 包括 xxx。

数据格式如下：

3.4 数据清洗

3.4.1 数据清洗规则

1. 数据质量（只要对的路径，只要对的 api）
2. 工具质量（太长的不要）
3. 不需要

3.4.2 ToolBench 数据清洗

基于历史的 API 工具调用结果，GPT-3.5 最后会得出一个”Final Answer“即正面回答用户需求，或者 GPT-3.5 会选择“Give Up”，即根据 API 工具调用的结果无法完成任务。而在 API 路径的寻找过程中，GPT-3.5 也会不断遇到 API 工具错误的问题，或者是工具调用成功，但是不符合用户需求的情况，因此我们也需要对 GPT-3.5 得到的调用路径进行修剪，来删除错误的 API 工具节点，仅保留成功调用并获取正确答案的 API 节点路径。

经过数据分析，在其中的工具轨迹数据中，共有 xxx 条，包含 xxx 词工具调用尝试，其中 xxx 次为失败调用，xxx 次为成功调用。而在所有的工具调用数据中，得出“Final Answer”的有 xx 条，占 xx%。选择“Give Up”的有 xx 条，占 xx%.

3.5 图谱构建

工具图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 当中的设计[16] 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。因此，在该工具图谱中，权值的计算方式非常重要。边的权值应该有区分度，不然无法提供有效的信息。

除了对工具之间边上的权值进行定义，本文还提供了对于节点的权值定义，该权值定义为该节点的存活状态。

API 工具具有时效性和多变性，为了让工具图谱能够更好地表示最新的工具状态，我们会根据调用历史对边和点的权值进行动态的更新。

3.5.1　静态构建

静态构建图的过程相对简单，与 2-gram 语言模型类似。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，例如 $[\text{tool}_1, \ldots, \text{tool}_s, \ldots, \text{end}]$。为了简化操作，我们将“结束”视为一种普通工具。

节点 v_i 到 v_j 的转移权重 $w_{i,j}$ 计算如下：

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

节点 v_i 的可用性权重计算如下：

$$\text{可用性权重} = \frac{\text{调用成功次数}}{\text{总次数}} \quad (0 \leq \text{可用性权重} \leq 1)$$

3.5.2　动态更新

1. 对于新加入的 api 工具 2. 对于新产生的调用轨迹

工具是动态变化的，具有生命周期，有些可能会随着时间的推移不再被开发者维护。因此，图中边的权重需要及时调整，这使得静态构建无法应对。此外，通常情况下，工具使用轨迹的数据量并不充足，尤其是对于像 ChatGPT 上新出现的插件。在这种情况下，动态构建 G 变得至关重要。

图 G 可以通过静态方法初始化，或者初始化为一个完全连接的无信息图。随后，动态构建会在生成工具使用轨迹和更新图 G 之间进行迭代。轨迹生成的过程与第 3 节中所描述的相同。图 G 的更新尤为关键。由于可用的工具使用轨迹数量有限，因此需要对每条轨迹进行细致的检查。LLM 作为工具评估器，将整个轨迹作为输入，对每个使用的工具进行打分。打分范围为 $[-3, 3]$ 的离散整数。评估工具相当于评估图中被访问的节点。我们用 $\Delta_i^{(n)}$ 表示第 n 次迭代中节点 v_i 的评估分数。累积得分 $s_i^{(n)}$ 计算如下：

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

节点 v_i 到 v_j 的转移权重 $w_{i,j}^{(n)}$ 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，$\Delta w_{i,j}^{(n)}$ 是用于更新转移权重的归一化梯度：

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

函数 $f(x)$ 将累积分数映射到 $(0, +\infty)$：

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

其中 $f(x)$ 是累积分数与更新梯度之间的映射函数。α 和 β 是超参数，α 控制更新速度，β 在初始权重 $w_{i,j}^{(0)}$ 和动态构建学习得到的权重之间插值。

3.6 API 召回模型训练

通过对现有的通用向量模型进行测试，如 BGE，BCE，M3E 等，我们发现通用的向量模型在用户需求与工具匹配任务上具有一定提升空间。因此，本文提出了基于通用向量模型进行微调，提升模型在工具选择上的能力。

本文在训练数据构造部分提出了细粒度、高难度的正负样本构建策略，有效提升了模型的工具搜索能力。

3.6.1 训练数据构造

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。但是对于负样本的部分，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）构造一种是困难负样本构造（Hard Negative）。

简单负样本构造。对于简单负样本构造，我们直接选择了不同大类的 K 个工具作为负样本。

基于大语言模型的困难负样本构造。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。

如下图所示，（介绍为什么需要 tune，明明两个很不同的相似度很高，但是很一致的相似度很低）。

因此，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。

3.6.2　模型训练及超参数设置

在进行向量模型微调之前，选择合适的基座模型是一个关键步骤，这将直接影响到微调后模型的性能以及效率。在选择基座模型时，通常需要考虑到模型的规模、大小，模型支持的语言，模型的训练任务等。

基于上述原因，本工作在对比性能后选择了 BGE-m3 模型作为基础模型进行训练。理由如下：

本文分别使用简单负样本构造和困难负样本构造方法构造了 xxx 条数据和 xxx 条数据，分别对模型进行了微调。同时，在测试集中，我们除了使用从 ToolBench 中筛选得到的数据。

本工作基于 BGE-m3 进行了微调。实现过程中将 epoch 数量设置为 xx，学习率设置为 3e-5 进行学习。模型大小为 xxx，在 xxx 芯片上训练了 xxx 得到了结果。

3.7　实验结果

表 3.1　API 工具召回实验结果

Method	I1	I2	I3	Average
BCE	30.1	40.0	28.6	**32.9**
BGE-Small	42.3	45.6	33.2	**40.4**
ToolBench	46.5	49.1	38.9	**44.8**
Ours	**58.0**	**70.6**	**62.8**	**63.8**

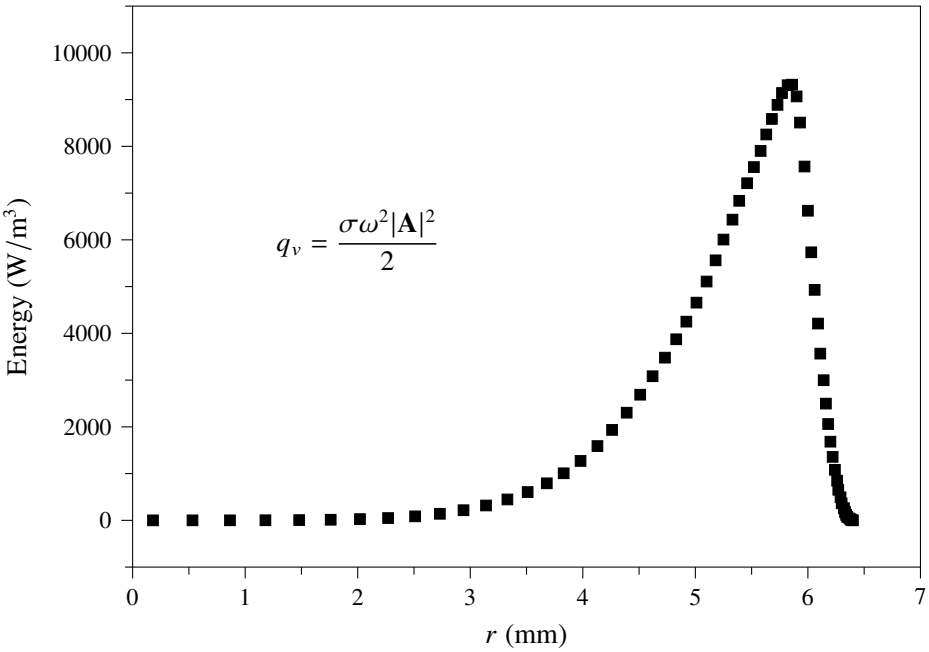


图 3.1 内热源沿径向的分布

Figure 3.1 Energy distribution along radial

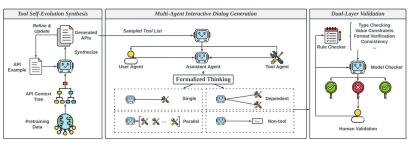


图 3.2 中文题图

Figure 3.2 English caption

3.8　插图

3.8.1　单个图形

3.8.2　多个图形

如果多个图形相互独立，并不共用一个图形计数器，那么用 `minipage` 或者 `parbox` 就可以，如图 3.3 与图 3.4。

图 3.3　并排第一个图

图 3.4　并排第二个图

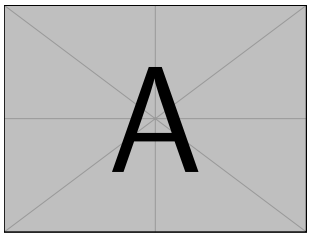
subcaption 宏包也提供了 subfigure 和 subtable 环境，如图 3.5。

(a) 校徽

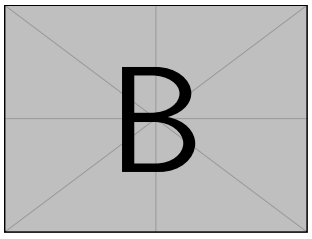
(b) 校名。注意这个图略矮些，subfigure 中同一行的子图在顶端对齐。

图 3.5　包含子图题的范例（使用 subfigure）

搭配 bicaption 宏包时，可以启用 `\subcaptionbox` 和 `\subcaption` 的双语变种 `\bisubcaptionbox` 和 `\bisubcaption`，如图 3.6 所示。



(a) R_3 = 1.5mm 时轴承的压力分布云图
(a) Pressure contour of bearing when R_3 = 1.5mm



(b) R_3 = 2.5mm 时轴承的压力分布云图
(b) Pressure contour of bearing when R_3 = 2.5mm

图 3.6　包含子图题的范例（使用 subcaptionbox）
Figure 3.6　Example with subcaptionbox

3.9 表格

3.9.1 基本表格

表 3.2 一个颇为标准的三线表[①]

Item		
Animal	Description	Price ($)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33
Armadillo	frozen	8.99

① 这个例子来自《Publication quality tables in LaTeX》（booktabs 宏包的文档）。这也是一个在表格中使用脚注的例子，请留意与 threeparttable 实现的效果有何不同。

第 4 章　知识图谱与大语言模型互增益的 API 编排方法

4.1　引言

在第三章中，我们介绍了基于工具调用轨迹构建的 API 知识图谱，包括有对调用轨迹数据的清洗、图谱的静动态构建算法、API 节点召回模型训练等部分。在本章，将会利用该图谱包含的 API 之间的依赖信息，集中在图谱利用和 API 路径编排的部分，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　问题定义

工具增强的大语言模型（LLM）通过结构化的自然语言文本与环境进行交互。我们提出了一种新的解决方案，结合了一个由 1.6 万个 API 构成的 API 图作为工具池，以优化 API 调用和任务执行的过程。具体的交互流程如下：

在接收到任务时，首先通过任务分解模块将任务分解为多个子任务，每个子任务将单独进行处理。任务分解的过程可以表示为：

$$T_{sub} = \text{Task\_decomposer}(T) \tag{1}$$

其中，T_{sub} 是分解后的子任务集合，T 是输入的原始任务。

对于每个子任务的 API 选择，采用 API 检索模型在 API 池中检索出前 k 个与子任务最相关的 API 作为初始节点。具体步骤可以形式化为：

$$A_{init} = \text{API\_retriever}(T_{sub}, q) \tag{2}$$

其中，A_{init} 表示初始节点的集合，T_{sub} 是 API 工具池，q 是当前子任务的查询。

随后，LLM 将基于初始节点进行迭代选择新的 API 节点并进行调用。这个选择过程不仅考虑当前任务的需求，还会记录选择过程中的调用结果，以便后续的回溯和调整：

$$a_s = P_\theta(C_s \cup o_s, t_s, A_{current}) \tag{3}$$

在上式中，$A_{current}$ 代表当前任务相关的 API 节点集合，而 a_s 是选择的 API 动作。

在所有子任务的 API 调用成功后，我们将每个子任务的输出和结果汇总到汇总器，最终得出一个综合的结论。汇总的过程可以表示为：

$$R_{final} = \text{Aggregator}(R_{sub}) \tag{4}$$

其中，R_{final} 是最终的结论，R_{sub} 是各子任务的结果集合。

为提升长期记忆能力，我们还存储过去的调用历史，并通过向量检索方法查找类似的查询和路径，从而辅助当前的决策过程。长期记忆的更新可以通过以下公式表示：

$$M_{long} = \text{Vector\_search}(H, q) \tag{5}$$

其中，M_{long} 表示长期记忆，H 是过去的调用历史，q 是当前查询。

此外，API 调用的过程经过简单配置参数后，可以直接进行调用，并对过长的响应进行压缩，以提高响应效率。这个过程可以表示为：

$$r_s = \text{API\_call}(a_s, \text{params}) \tag{6}$$

在这里，r_s 表示 API 的响应，a_s 是所选择的 API 动作，params 是配置的参数。

通过这样的设计，我们的基于大语言模型 Agent 和图谱的 API 编排和调用工具能够有效地处理复杂任务，优化 API 的调用流程和准确性。

4.3 整体框架

图 3.1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。

该方法由以下几个部分组成：

1. 任务分解模块：任务分解模块负责将复杂、模糊的用户需求分解为多个子任务，能够充分提高系统处理复杂任务的能力。
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。动态搜索能够提高系统的灵活性和可用性，自我反思机制能够通过格式化的反思提升系统的准确性，长短期记忆能够通过历史记录辅助当前决策。
3. 任务执行模块：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。

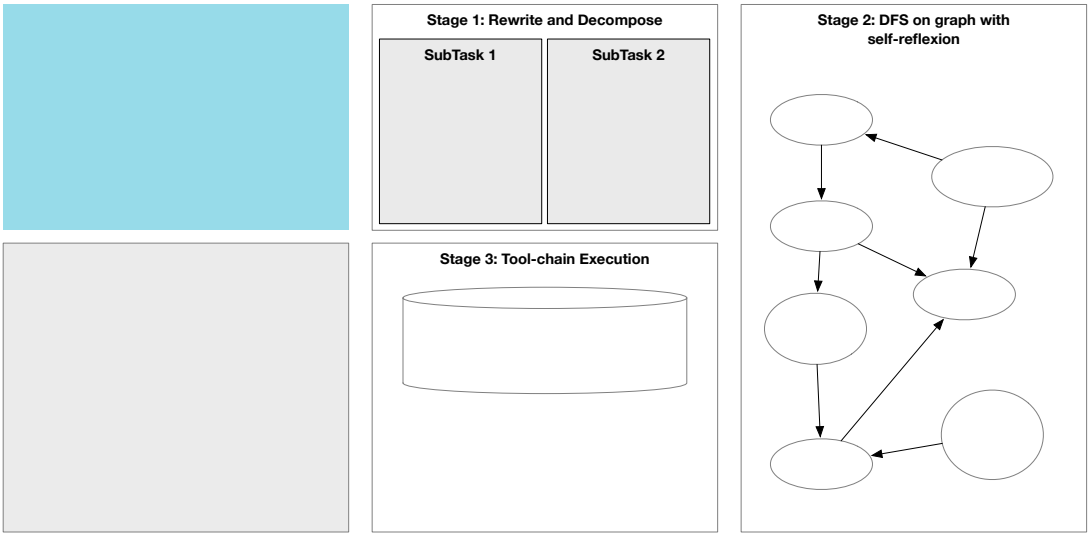


图 4.1 整体框架
Figure 4.1 Framework

4.4 算法实现

4.4.1 任务改写与子任务拆解模块

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分解、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。

任务分解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，任务分解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分解。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。

其次，改写器的作用是将分解后的子任务进一步精炼成更具体、明确的语句。这一过程借助于大语言模型强大的自然语言处理能力，通过对原始需求的语义理解和推理，将模糊的任务表达转化为更加清晰、具体的指令。例如，用户可能提出“帮我分析特斯拉的股票数据并生成趋势报告”的需求。改写器会将其分解为多个步骤：首先获取股票数据，然后分析数据中的关键趋势，最后生成一份详细报告。通过这一改写过程，系统可以更好地执行每一步任务，通过调用查询数据的工具 API 获得相关信息，并在获取股票数据后，通过大语言模型自身的分析功能生成分析报告。

此外，任务分解模块能够显著提升大语言模型在多工具和多类别场景中的应用能力。通过将复杂任务拆解为多个相互独立的子任务，系统能够并行执行这些子任务，并分别调用不同的工具或 API 来完成每一步。子任务中包含“任务类别”，通过在任务分解模块中划分好任务的类别，可以有效减少后续动态工具搜索时的搜索空间，提升搜索的准确性和速度。这种设计允许各个子任务同时进行，显著提高任务执行效率。

总的来说，基于大语言模型的任务分解模块通过分解复杂任务、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。

An Example for Response Generation with GPT-4

Instruction Prompt: You are currently in the response generation stage. You are given the output of selected tools that can be potentially used to solve the sub-task. Your goal is to generate a response to the user question based on the results from tools and your internal knowledge.

User Question: I would like to know the value of 5 ounces of gold plus 1 million AMZN stocks in CNY.

Return from Tool Calling: The stock price of AMZN is 130 USD, XAU USD is 1919.7605807522993, and the exchange rate between USD and YNC is 7.24.

Output: Total value in CNY: Total value = Value of gold in CNY + Value of stocks in CNY = 69,495.87 + 941,200,000 = 941,269,495.87 CNY. Therefore, the combined value of 5 ounces of gold and 1 million Amazon stocks in Chinese Yuan is approximately 941,269,495.87 CNY.

4.4.2　基于深度优先遍历的动态搜索算法

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，该算法的优点在于能够防止错误传播，并对整体工具行动空间进行更深入的探索。

我们提出的基于深度优先的寻路算法的流程图如下所示：

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，我们利用基于语义相似度的 API 召回器，得到与用户需求在语义上最相似的一组工具，让大语言模型进行选择，最终确定初始节点，并从该节点开始进行后续探索。

其次，我们在图上构建一棵深度优先搜索树，允许模型在图上不断选择新的节点并评估每一条推理路径。具体逻辑如下：对于每个选择的节点，我们调用该节点并获取对应的 API 响应内容，然后将已经调用的工具轨迹、解析后的 API 响应、用户需求等提供给大语言模型，让模型选择下一步操作。下一步操作可以是继续选择节点，

也可以是回溯到上一节点。

如果模型选择了下一步操作，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。对于邻居节点的选择，我们会获取到当前节点权值最大的 K 个节点作为下一步操作。如果邻居节点不足 5 个，我们将重新调用 API 召回器，以补全 5 个选择的选项。模型会根据当前状态继续迭代选择，直到结束选择或放弃。

如果模型选择回溯到上一节点，我们将在短期记忆模块中添加回溯步骤，并恢复到上一个节点的状态。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组，该任务不可行，即放弃任务执行。

4.4.3　自我反思机制

本文构建的自我反思框架会在动态寻路算法选择“放弃”或者评判器将该路径判定为“失败”的情况下被激活。在自我反思框架中，我们首先需要进行“反思”，即根据当前编排得到的工具调用路径，识别未能成功满足用户需求的原因。具体而言，当动态寻路算法的选择器决定“放弃”某条路径时，会提供该路径放弃的具体原因。这些“放弃原因”可作为反思的依据，以指导模型在后续的路径选择中作出更为合理的决策。此外，在动态寻路算法得出一个解决方案后，若经过评估器的评估发现该路径未能充分满足用户需求，则会引用评估器所提供的失败理由。评估器在评估失败时会对该路径及总结器的最终回答提供一个等级，根据失败的等级我们可以选择以下两种不同的重新寻路算法：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，我们将持续记录检索过程中的每一步输入、输出及上下文信息。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**：在评判器认为路径“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

算法 4.1 图谱节点选择算法

```
Data: 用户需求 user
Result: 选择合适的工具
// 选择初始节点;
initialNodes ← API(user);
selectedNode ← LLM(initialNodes);
// 初始化深度优先搜索树;
dfsTree ← new Tree();
dfsTree.addNode(selectedNode);
while true do
    response ← API(selectedNode);
    // 提供信息给大语言模型;
     ← {工具轨迹, API 响应, 用户需求};
    nextAction ← LLM();
    if nextAction == "选择下一节点" then
        topNeighbors ← (selectedNode);
        if length(topNeighbors) < 5 then
            topNeighbors ← API(selectedNode);
        end
        selectedNode ← LLM(topNeighbors);
        dfsTree.addNode(selectedNode);
    else
        // 回溯逻辑;
        .add(dfsTree.currentNode());
        selectedNode ← dfsTree.backtrack();
        if selectedNode == dfsTree.root() && 需要继续回溯() then
            放弃任务 ();
            break;
        end
    end
    if 模型结束选择() then
        break;
    end
end
```

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。尽管本文提出的基于深度优先遍历的算法可以通过回溯来避免错误传播的问题，但我们也应尽可能地避免选择错误和无关的工具，这样会增加寻路时间，也会使算法面临更多的不稳定性。

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深入理解和逐步优化，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。

4.4.4　工具调用路径长短期记忆框架

4.4.4.1　长短期记忆框架概述

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。长期记忆部分主要有记忆存储、召回和利用三个部分，围绕着如何进一步利用过去成功推理的经验来辅助后续的推理和规划过程。

4.4.4.2　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、轨迹路径信息等。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。

4.4.4.3　长期记忆

长期记忆是与短期记忆相对的概念，长期记忆会固定地存在数据库中，并且随着调用次数增多而积累。长期记忆记录的是最终形成的工具调用链以及结果，每次在系统进行推理时都会从长期记忆库中进行搜索，从而利用历史经验知识来辅助模型的推理。

长期记忆的具体实现方式如下，包括记忆的添加、删除、修改和查询四个部分。

1. **记忆添加**

   记忆增加是该框架中最基础的功能。首先，除了初始记忆批量添加的阶段，其他的记忆添加都是在生成了新的推理路径时进行。我们先对新生成的推理路径进行筛选，通过一个评判器来对推理路径判断质量。如果评判器认定为“失败”

或“不确定”，那么就舍弃当前的推理路径。这一步的判别是为了保证记忆的质量，不存储有失败风险的路径在记忆中。随后，我们将用户的需求与推理路径形成一个二元组，将用户需求输入向量模型转为嵌入向量，将推理路径作为文本格式存储，方便计算和检索记忆。

2. **记忆修改**

   对于同一个任务，系统可能会生成不同的推理路径。在这种情况下，我们会用新的记忆来替换旧的记忆，对记忆做出及时的更新。在记忆修改时，我们只需要对推理路径部分进行修改，并保存更新后的路径在数据库中，而不需要对用户需求的向量部分进行修改。

3. **记忆删除**

   由于工具节点的状态是随时变化的，比如开发者停止维护了某个工具，或者是某个工具 API 出现故障等等，因此有的时候我们会删除一些工具节点或修改节点状态为“失效”。在这种情况下，我们除了在工具图谱上进行修改，也应该同时删除有关的记忆，避免对模型造成困扰。记忆删除的逻辑较为清晰，即当某工具被删除或失效时，在记忆库中匹配所有包含该工具的路径并删除对应的记忆。

4. **记忆查询**

   在有新的用户需求时，系统会首先将用户需求转为向量形式，然后在向量数据库中通过相似度检索算法查找到与当前需求最相似的 K 个历史需求以及对应的工具轨迹，即搜索得到的记忆。

4.4.5 记忆搜索与召回

4.4.6 记忆利用

（可以做消融实验）

4.4.7 工具调用模块

4.4.7.1 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生成请求体，然后通过工具规划流程确定请求体中的参数，最后结合用户需求填充请求体中的参数值。在生成请求体后，我们通过 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，我们首先检查响应内容的长度，以决定是否需要对 API 响应进行压缩。最后，经过压缩的 API 响应将被存储在路径规划智能体的“短期记忆”中，即模型的上下文，以辅助模型生成最终结果。

4.4.8　工具调用模块

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。

4.4.8.1　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。具体步骤包括：

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。

（todo，做实验）

4.5 实验与评估

4.5.1 测试数据集

ToolBench。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 工具的推理轨迹数据。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。

API-Bank。API-Bank**Li2023**是另一个公开的工具调用数据集，作者们针对模型工具调用的检索、规划能力的评估精心构建了测试数据，其中包含 73 个 API 工具，并对 314 个工具调用进行了标注。本工作从中抽取了 300 条数据作为测试集。

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。

4.5.1.1　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 5 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。

4.5.1.2　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。

4.5.2 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：

$$PR = \frac{\#(\text{Solved})}{\#(\text{Solved}) + \#(\text{Unsolved})}. \tag{4.1}$$

- **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：

$$WR = \frac{\#(\text{Won})}{\#(\text{Won}) + \#(\text{Lost}) + \#(\text{Tie})}. \tag{4.2}$$

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。

4.5.3　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-4 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。

4.5.4　实验结果

实验设置

4.5.5　错误分析

4.6　本章小结

第 5 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

5.1 系统需求分析

我们采用面向对象的需求分析方法，绘制了如图所示的系统用例图。

“如图 5.1 所示”等。该页空白不够排写该图整体时，则可将其后文字部分提前排写，将图移到次页。

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

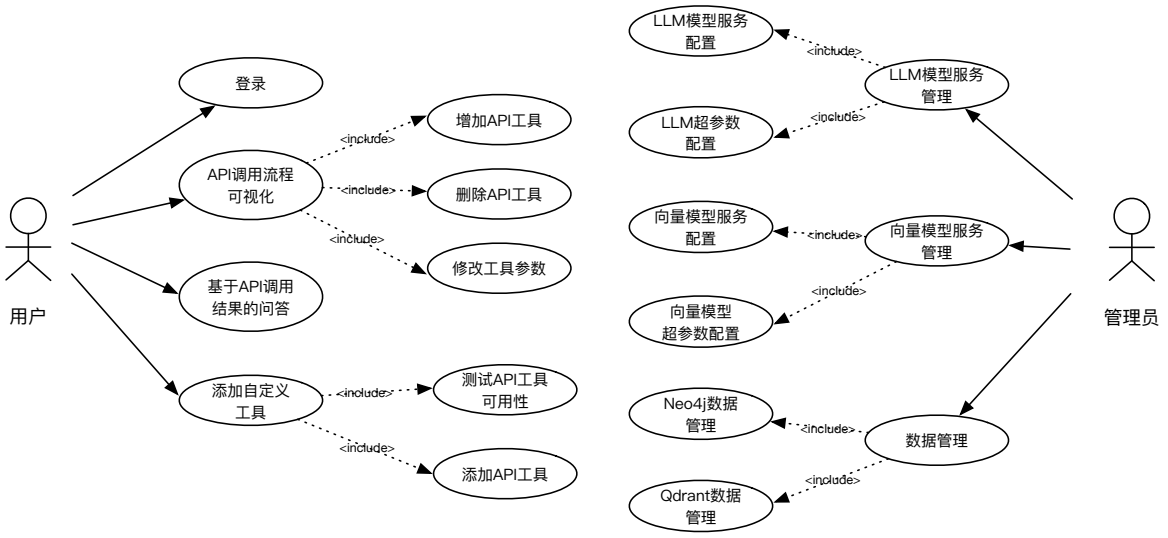


图 5.1　系统用例图
Figure 5.1　System Usecase Diagram

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。

5.2　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。

5.2.1　交互方案设计

5.2.2　系统架构设计

本系统的架构将会从部署架构和软件架构来进行说明。

部署架构。

软件架构。

5.2.3 数据管理层

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

5.2.4 API 编排层

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。

该层的主要流程为：首先通过拆解模块将任务拆解为独立的子任务，然后针对每个子任务，检索类似任务作为参考，并检索相关 API 信息，构建提示词，基于大模型的推理能力和上下文生成结构化的 API 流程。

5.2.5 API 调用层

API 调用层主要负责根据 API 编排层生成的调用流程，获取调用参数并执行 API 调用。主要涉及到以下部分：

- **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。
- **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。
- **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。

5.2.6　UI 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。

UI 层代码是基于 streamlit 代码库实现的。

5.2.7　数据库设计

画 er 图。

5.2.8　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。

5.2.9　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。

5.2.10　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。

5.2.11 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。

5.2.12 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。

5.2.13 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。

5.2.14 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。

5.3 系统实现

技术选型。

5.4 系统展示

（此处为系统截图展示）

5.5 本章小结

第 6 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。

6.1 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。

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。

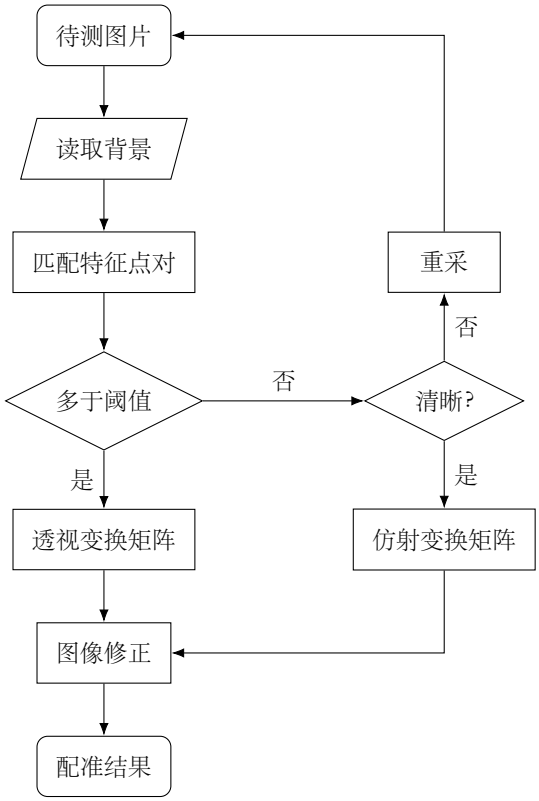


图 B.1 绘制流程图效果
Figure B.1 Flow chart

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.